TradeStation®

Learning
EasyLanguage
Functions and Reserved Words

TradeStation®

Important Information and Disclaimer:

All support, education and training services and materials provided by TradeStation are for informational affiliate.

This material may also discuss in detail how TradeStation is designed to help you develop, test and implement trading strategies. However, TradeStation does not provide or suggest trading strategies. We offer you unique tools to help you design your own strategies and look at how they could have performed in the past. While we believe this is very valuable information, we caution you that simulated past performance of a trading strategy is no guarantee of its future performance or success. We also do not recommend or solicit the purchase or sale of any particular securities or derivative products. Any symbols referenced are used only for the purposes of the demonstration, as an example—not a recommendation.

Finally, this material may discuss automated electronic order placement and execution. Please note that even though TradeStation has been designed to automate your trading strategies and deliver timely order placement, routing and execution, these things, as well as access to the system itself, may at times be delayed or even fail due to market volatility, quote delays, system and software errors, Internet traffic, outages and other factors.

TradeStation Group, Inc. Affiliates: All proprietary technology in TradeStation is owned by TradeStation Technologies, Inc. Equities, equities options, and commodity futures products and services are offered by TradeStation Securities, Inc. (Member NYSE, FINRA, CME and SIPC). TradeStation Securities, Inc.'s SIPC coverage is available only for equities and equities options accounts.

Table Of Contents

Functions

@Delta (Function)

Disclaimer

The @Delta function calculates the risk value Delta of an option or position. This function is designed primarily for use in OptionStation.

Syntax

```
@Delta(Daysleft, StrikePr, AssetPr, Rate100, Volty100, PutCall)
```

Returns (Double)

A numeric value representing the Delta value of an option.

Parameters

Name	Type	Description
DaysLeft	Numeric	Sets the number of calendar days left for the option.
StrikePr	Numeric	Sets the strike price of option.
AssetPr	Numeric	Specifies the price of the underlying asset
Rate100	Numeric	Sets the risk-free annual interest rate, in percent.
Volty100	Numeric	Sets the volatility of annual return, in percent, of the underlying asset.
PutCall	Numeric	Sets if it is a Put or Call option. Put or 2 = Puts; Call or 3 = Calls.

Remarks

The Delta of an option (on non-dividend paying stock) equals the option price sensitivity vs. stock price.

With the input parameter DaysLeft you can use the date functions DaysToExpiration or Next3rdFriday.

With the input parameter Rate100, you can pass in a constant or attach an appropriate index as Data 2 and pass in its price.

With the input parameter Volty100 you can pass in a constant or use one of the volatility functions - ImpliedVolatility or VolatilityStdDev.

Example

```
Plot1( @Delta( DaysToExpiration(3, 2007), 34.90, Close, Rate100,  Volty100, PutCall
), "Delta" ) ;
```

@Gamma (Function)

Disclaimer

The @Gamma function calculates the risk value Gamma of an option or position. This function is designed primarily for use in OptionStation.

Syntax

```
@Gamma(Daysleft,StrikePr,AssetPr,Rate100,Volty100,PutCall)
```

Returns (Double)

A numeric value representing the Gamma value of an option.

Parameters

Name	Type	Description
DaysLeft	Numeric	Sets the number of calendar days left for the option.
StrikePr	Numeric	Sets the strike price of option.
AssetPr	Numeric	Specifies the price of the underlying asset
Rate100	Numeric	Sets the risk-free annual interest rate, in percent.
Volty100	Numeric	Sets the volatility of annual return, in percent, of the underlying asset.
PutCall	Numeric	Sets if it is a Put or Call option. Put or 2 = Puts; Call or 3 = Calls.

Remarks

The Gamma of an option (on non-dividend paying stock) equals the Delta sensitivity vs. stock price.

With the input parameter DaysLeft you can use the date functions DaysToExpiration or Next3rdFriday.

With the input parameter Rate100, you can pass in a constant or attach an appropriate index as Data 2 and pass in its price.

With the input parameter Volty100 you can pass in a constant or use one of the volatility functions - ImpliedVolatility or VolatilityStdDev.

Example

```
Plot1( @Gamma( DaysToExpiration(3, 2007), 34.90, Close, Rate100,  Volty100, PutCall
), "Gamma" ) ;
```

@Theta (Function)

Disclaimer

The @Theta function calculates the risk value Theta of an option or position. This function is designed primarily for use in OptionStation.

Syntax

```
@Theta(Daysleft, StrikePr, AssetPr, Rate100, Volty100, PutCall)
```

Returns (Double)

A numeric value representing the Theta value of an option.

Parameters

Name	Type	Description
DaysLeft	Numeric	Sets the number of calendar days left for the option.
StrikePr	Numeric	Sets the strike price of option.
AssetPr	Numeric	Specifies the price of the underlying asset
Rate100	Numeric	Sets the risk-free annual interest rate, in percent.
Volty100	Numeric	Sets the volatility of annual return, in percent, of the underlying asset.
PutCall	Numeric	Sets if it is a Put or Call option. Put or 2 = Puts; Call or 3 = Calls.

Remarks

The Theta of an option (on non-dividend paying stock) equals the option price sensitivity vs .time.

With the input parameter DaysLeft you can use the date functions DaysToExpiration or Next3rdFriday.

With the input parameter Rate100, you can pass in a constant or attach an appropriate index as Data 2 and pass in its price.

With the input parameter Volty100 you can pass in a constant or use one of the volatility functions ImpliedVolatility or VolatilityStdDev.

Example

```
Plot1( @Theta( DaysToExpiration(3, 2007), 34.90, Close, Rate100,  Volty100, PutCall
), "Delta" ) ;
```

@Vega (Function)

Disclaimer

The @Vega function calculates the risk value Gamma of an option or position. This function is designed primarily for use in OptionStation.

Syntax

```
@Vega(Daysleft,StrikePr,AssetPr,Rate100,Volty100,PutCall)
```

Returns (Double)

A numeric value representing the Vega value of an option.

Parameters

Name	Type	Description
DaysLeft	Numeric	Sets the number of calendar days left for the option.
StrikePr	Numeric	Sets the strike price of option.
AssetPr	Numeric	Specifies the price of the underlying asset
Rate100	Numeric	Sets the risk-free annual interest rate, in percent.
Volty100	Numeric	Sets the volatility of annual return, in percent, of the underlying asset.
PutCall	Numeric	Sets if it is a Put or Call option. Put or 2 = Puts; Call or 3 = Calls.

Remarks

The Vega of an option (on non-dividend paying stock) equals the option price sensitivity vs. volatility.

With the input parameter DaysLeft you can use the date functions DaysToExpiration or Next3rdFriday.

With the input parameter Rate100, you can pass in a constant or attach an appropriate index as Data 2 and pass in its price.

With the input parameter Volty100 you can pass in a constant or use one of the volatility functions - ImpliedVolatility or VolatilityStdDev.

Example

```
Plot1( @Vega( DaysToExpiration(3, 2007), 34.90, Close, Rate100,  Volty100, PutCall
), "Gamma" ) ;
```

AB_AddCellRange (Function)

Disclaimer

The AB_AddCellRange function adds/plots a range of cells from the high price to the low price of the ActivityBar for each interval.

Syntax

```
AB_AddCellRange(RangeHi, RangeLo, Side, CellGroupLabel, CellGroupColor,
    CellGroupValue)
```

Returns (Integer)

This function returns 1 if the drawing of cells was successful, otherwise, -1 if unable to add cells.

Parameters

Name	Type	Description
RangeHi	Numeric	Sets the highest price level of the column.
RangeLo	Numeric	Sets the lowest price level of the column.
Side	Numeric	Sets the side of the bar on which to add the column of cells. Use the reserved words RightSide (1) or LeftSide (-1).
CellGroupLabel	String	Sets a string character that will be placed inside each cell.
CellGroupColor	Numeric	Sets the coloration of the cells for that particular column of cells.
CellGroupValue	Numeric	Sets a value that is assigned to each cell that is added.

Remarks

The function will add the cells to either the right of left of the bar depending on the Side input. You can use either LeftSide or RightSide. The letter specified in the input CellGroupLabel will be placed in each one of the cells. The cells will be drawn using the CellGroupColor color. The cells will have the CellGroupValue assigned to them.

If RangeHi < RangeLo, the function will not add any cells and return a –1.

Examples

The first example adds a number of blue cells with a "+", from the High of the ActivityBar to the Low of the ActivityBar. The assigned value of the cells will be 0.

```
AB_AddCellRange (High of ActivityData, Low of ActivityData, RightSide, "+", Blue,
0);
```

This second example could be used if we wanted to conditionally add multiple green cells with a "U" to the current bar, from the High of the ActivityData to the Low of the ActivityData, if the Close of the ActivityData is greater than the Close of the previous ActivityData bar. Below is the EasyLanguage for this concept:

```
If Close of ActivityData > Close[1] of ActivityData Then
 Value1 = AB_AddCellRange(High of ActivityData, Low of ActivityData, RightSide, "U",
Green, 0);
```

AB_AverageCells (Function)

Disclaimer

The `AB_AverageCells` functoin returns the average number of ActivityBar cells per row for the current bar.

Syntax

```
AB_AverageCells(Side);
```

Returns (Double)

The average number of ActivityBar cells on the specified `Side`(s) of the current bar, otherwise, returns a 0 if no cells are found.

Parameters

Name	**Type**	**Description**
`Side`	Numeric	Sets the side of the ActivityBar that will be averaged; options are `LeftSide`, `RightSide,` or 2 for both sides.

Remarks

The value returned by the function can be calculated over the entire ActivityBar, or it can be restricted to either side with the `Side` input.

Example

Assigns to `Value1` the average number of cells per row on both sides of the current bar:

```
Value1 = AB_AverageCells(2);
```

Assigns to `Value2` the average number of cells per row on the right side of the current bar:

```
Value2 = AB_AverageCells(RightSide);
```

AB_AveragePrice (Function)

Disclaimer

The AB_AveragePrice function calculates the average price of ActivityBar cells.

Syntax

```
AB_AveragePrice(Side)
```

Returns (Double)

The average price of the ActivityBar cells on a particular side or over the entire bar.

Parameters

Name	Type	Description
Side	Numeric	Sets the side of the ActivityBar that will be averaged; options are LeftSide, RightSide and 2 for both sides.

Remarks

The value returned by the function can be calculated over the entire ActivityBar, or it can be restricted to either side with the Side input. The Average is based on price and number of cells in each row.

Examples

Assigns to Value1 the average price for the cells on both sides of the ActivityBar:

```
Value1 = AB_AveragePrice(2);
```

Assigns to Value2 the average price for the cells on the right side of the ActivityBar:

```
Value2 = AB_AveragePrice(RightSide);
```

See Also

AB_AverageCells

AB_CellCount (Function)

Disclaimer

The `AB_CellCount` function counts the number of cells on one or both sides of an ActivityBar.

Syntax

```
AB_CellCount(Side)
```

Returns (Integer)

The number of ActivityBar cells on the specified `Side`(s) of the current bar.

Parameters

Name	Type	Description
`Side`	Numeric	Sets the side of the ActivityBar on which the cells will be counted; options are `LeftSide`, `RightSide` and 2 for both sides.

Example

Assigns `Value1` the number of cells that are drawn on both sides of the current bar:

```
Value1 = AB_CellCount(2);
```

Assigns `Value2` the average number of cells on the right side of the bars of the last 3 bars:

```
Value2 = Average(AB_CellCount(RightSide), 3);
```

AB_Median (Function)

Disclaimer

The AB_Median function calculates and returns the median value for the ActivityBar cells of the current bar.

Syntax

```
AB_Median(Side)
```

Returns (Double)

The median value for ActivityBar cells on the specified Side(s) of the current bar.

Parameters

Name	Type	Description
Side	Numeric	Sets the side of the ActivityBar on which the median will be calculated; options are LeftSide, RightSide and 2 for both sides.

Remarks

The median refers to the middle value in the cell distribution, which is between an equal number of values on one or both sides of the ActivityBar. The Side Input determines if the calculation will take place over the whole bar or just on a specific side.

Example

Assigns to Value1 the median price of the cells drawn on both sides of the current bar:

```
Value1 = AB_Median(2);
```

Assigns to Value2 the median price of the cells drawn on the left side of the current bar:

```
Value2 = AB_Median(LeftSide);
```

See Also

AB_Mode, AB_StdDev

AB_Mode (Function)

Disclaimer

The AB_Mode function returns the number of cells and price in the Mode row of the current bar.

Syntax

```
AB_Mode(Side, Type, oModeCount, oModePrice)
```

Returns (Integer)

The oModeCount and oModePrice output parameters return the number of cells and price in the mode row. The AB_Mode function itself returns a value of 1 if successful and -1 otherwise.

Parameters

Name	Type	Description
Side	Numeric	Sets the side of the ActivityBar on which the Mode will be calculated; options are LeftSide, RightSide and 2 for both sides.
Type	Numeric	Sets either the largest or smallest mode to indicate (>= 0 for largest, < 0 for smallest).
oModeCount	Numeric	Outputs the mode count in the mode row.
oModePrice	Numeric	Outputs the mode price in the mode row.

Remarks

The Mode refers to the ActivityBar row that contains the greatest number of cells over the entire bar or on a particular side. AB_Mode refers to the number of cells in the Mode row and the price value of the row in which the Mode occurred. The Side Input determines if the calculation will take place over the whole bar or just on a specific side.

Examples

Assigns to Value2 the number of cells in the largest modal row and to Value3 the cell price on both sides of the current bar:

```
vars: oModeCount(0), oModePrice(0);
Value1 = AB_Mode(2, 1, oModeCount, oModePrice);
Value2 = oModeCount;
Value3 = oModePrice;
```

See Also

AB_Median, AB_StdDev

AB_NextColor (Function)

Disclaimer

The `AB_NextColor` series function returns the color of ActivityBar cells based on the minute interval count within the current bar.

Syntax

```
AB_NextColor(MinuteInterval)
```

Returns (Integer)

The RGB color that is associated with ActivityBar cells based on the current `MinuteInterval` period.

Parameters

Name	Type	Description
`MinuteInterval`	Numeric	Sets the number of minutes that make up each cell color interval.

Remarks

If the total number of intervals for a given ActivityBar exceeds 16 (the number of available colors), the reserved word return is reset to 1 and the reserved word will begin to increment again from that initial value. The qualifier, ‘of ActivityData’ is usually added to the reference of this reserved word so that it is called on each and every ActivityData bar, as opposed to just once for each new ActivityBar.

Example

Assuming a 60-minute ActivityBar with a 10-minute ActivityData interval, `Value1` is assigned a new color every 10 minutes for the cells that are added to the ActivityBar during that period. This results in a total of 6 color changes throughout the ActivityBar.

```
Value1 = AB_NextColor(10);
```

Assuming a 20-minute ActivityBar, with a 1-minute ActivityData interval, `Value2` is assigned a new color every 5 minutes for the cells that are added to the ActivityBar during that period. This results in a total of 4 color changes throughout the ActivityBar.

```
Value2 = AB_NextColor(5);
```

Additional Example

The EasyLanguage that would allow for the Minute Interval to be set as an input in an ActivityBar study that adds a cell, containing an "X", on the right side, at each price of the ActivityData, is displayed below. The default used in this example is a 10-minute interval.

```
Inputs: MinuteInt(10);
Variables: Color(0);
Color = AB_NextColor(MinuteInt) of ActivityData;
AB_AddCell(Close of ActivityData, RightSide, "X", Color, 0);
```

AB_NextLabel (Function)

Disclaimer

The AB_NextLabel series function determines the label that is placed within the cells of an ActivityBar, based on a user-defined interval.

Syntax

```
AB_NextLabel(MinuteInterval)
```

Returns (String)

The text character that is associated with ActivityBar cells based on the current MinuteInterval period.

Parameters

Name	Type	Description
MinuteInterval	Numeric	Sets the number of minutes that make up each cell label interval.

Remarks

The ActivityBar characters alternate in order from A to Z. Following the letter Z, numerical values from 1 to 9 are used. If all available alphabetical and numerical characters are used, the reserved word will return an asterisk "*", until the reserved word is reset on the next ActivityBar. Thus, there can be a total of 36 distinct intervals.

The MinuteInterval Input must be equal to or greater than the Bar Interval of the ActivityData. If the MinuteInterval Input exceeds the Bar Interval of the ActivityBar, only a single alphabetical character will be displayed throughout the entire ActivityBar. The qualifier, 'of ActivityData' is usually added to the reference of this reserved word so that it is called on each and every ActivityData bar, as opposed to just once for each new ActivityBar.

This reserved word does not affect the color of the cells, only the string contents.

Examples

Assuming a 60 minute ActivityBar with a 10 minute ActivityData interval:

```
Value1 = AB_NextLabel(10);
```

Value1 represents a character every 10 minutes for the cells that are added to the ActivityBar during that period. This results in a total of 6 characters used in the cells throughout the ActivityBar.

In addition, assuming a 20 minute ActivityBar, with a 1 minute ActivityData interval:

```
Value2 = AB_NextLabel(5);
```

Value2 represents a new character every 5 minutes for the cells that are added to the ActivityBar during that period. This results in a total of 4 characters used in the cells throughout the ActivityBar.

Additional Example

The EasyLanguage that would allow for the Minute Interval to be set as an Input in an ActivityBar study that adds a Red cell, on the right side, containing an interval specific character, at each price of the ActivityData, is displayed below. The default used in this example is a 10-minute interval.

```
Inputs: MinuteInt(10);
Variables: Letter("");
Letter = AB_NextLabel(MinuteInt) of ActivityData;
AB_AddCell(Close of ActivityData, RightSide, Letter, Red, 0);
```

See Also

AB_NextColor

AB_RowHeightCalc (Function)

Disclaimer

The AB_RowHeightCalc function calculates and returns the row height to be used for an ActivityBar.

Syntax

```
AB_RowHeightCalc(ApproxNumRows, RangeAvgLength)
```

Returns (Double)

The row height of an ActivityBar, or .1 if either ApproxNumRows or RangeAvgLength is set to zero.

Parameters

Name	Type	Description
ApproxNumRows	Numeric	Sets the approximate number of cell rows that are desired for an ActivityBar. The normal range for this Input is between 5 and 25.
RangeAvgLength	Numeric	Sets the number of bars to consider in the calculation of the average range.

Remarks

AB_RowHeightCalc uses a calculation of the average range, divided by the ApproxNumRows Input to determine what the row height should be set to. As a result of this calculation, the number of rows on each ActivityBar will be likely to vary. If the above calculation results in a row height value that is smaller than the minimum movement of the symbol, the minimum movement value is returned.

Examples

Approximates that there will be 10 rows for each ActivityBar, based on the average range over the last 3 bars:

```
AB_RowHeightCalc(10, 3)
```

In this second example, the AB_RowHeightCalc value, based on an approximation of 15 rows and using a 3 bar average range, is used with the AB_SetRowHeight reserved word to set the row height for an ActivityBar study.

```
AB_SetRowHeight(AB_RowHeightCalc(15, 3));
```

AB_StdDev (Function)

Disclaimer

The AB_StdDev function calculates the standard deviation of the ActivityBar cells.

Syntax

```
AB_StdDev(NumDevs, Side)
```

Returns (Double)

The standard deviation of the ActivityBar cells on specified sides.

Parameters

Name	Type	Description
NumDevs	Numeric	Sets the number of standard deviations to calculate.
Side	Numeric	Sets the side of the ActivityBar on which the Standard Deviation will be calculated; options are LeftSide, RightSide and 2 for both sides.

Remarks

Standard Deviation is defined as the statistical measure of the dispersion or variation in a distribution, equal to the square root of the arithmetic mean of the squares of the deviations from the mean. The Side Input determines if the calculation will take place over the whole bar or just on a specific side.

You can specify how many standard deviations to calculate. If no parameter is used, one standard deviation is calculated.

Examples

Assigns to Value1 the first standard deviation of the cell prices on both sides of the current bar:

```
Value1 = AB_StdDev(1, 2);
```

Assigns to Value2 the two standard deviation of the cells drawn on the left of the current bar:

```
Value2 = AB_StdDev(2, LeftSide);
```

See Also

AB_Mode, AB_Median

AbsoluteBreadth (Function)

Disclaimer

The AbsoluteBreadth function calculates the absolute difference between advancing versus declining data set.

Syntax

```
AbsoluteBreadth(AdvIssues,DecIssues)
```

Returns (Integer)

A positive number that represents the difference between two values for the current bar.

Parameters

Name	Type	Description
AdvIssues	Numeric	Specifies a the advancing issues price data series. Normally the Close of Data1 referencing the close of an advancing issues symbol.
DecIssues	Numeric	Specifies a the declining issues price data series. Normally the Close of Data2 referencing the close of a declining symbol.

Remarks

Traditionally, this calculation has been used with NYSE advancing issues and declining issues data to provide a gauge of market activity and volatility without regard to price direction, but both the NASDAQ and AMEX stock exchanges also provide this information.

You will need to create a multidata chart with both advancing issues and declining issues in the chart to use this function.

The AdvIssues input parameter will need to reference the close of the advancing issues symbol, and DecIssues will need to reference the close of the declining issues symbol.

Advancing issues and Declining issues symbols for the three major stock exchanges can be found using the symbol lookup feature. For more information, see Using the Symbol Lookup Feature.

The TradeStation Data Network symbols for NYSE advancing and declining issues are $ADV and $DECL.

Example

Assigns the AbsoluteBreadth value to Value1, where the advancing issue symbol is in Data1 and the declining issue symbol is in Data2, on a daily chart; then plots Value1:

```
Value1 = AbsoluteBreadth(Close of Data1, Close of Data2);
Plot1(Value1, "AbsBrdth");
```

AccumDist (Function)

Disclaimer

The `AccumDist` series function calculates the accumulation distribution of a security by adding the day's volume (or ticks) to a cumulative total when the price closes up, and subtracting the day's volume from the cumulative total when the price closes down.

Syntax

```
AccumDist(AnyVol)
```

Returns (Double)

A positive or negative number for the current bar representing the accumulation of up vs down volume (ticks).

Parameters

Name	Type	Description
`AnyVol`	Numeric	Specifies the volume (or ticks) for one of the symbols in the chart.

Remarks

Traditionally, this calculation attempts to show if volume is flowing into or out of a security. When the security closes higher than the open, all of the day's volume is considered up volume. When the security closes lower than the open, all of the day's volume is considered down volume.

The use of Trade Volume or Tick Count depends on the **For Volume, Use**: setting for the specified symbol.

This is a cumulative calculation type function, meaning that the current bar’s value for this function is based on the values of all previous bars.

Example

Assigns the accumulation distribution value to `Value1`, using the Volume of Data1; then plots `Value1:`

```
Value1 = AccumDist(Volume);
Plot1(Value1, "AccDist");
```

AccumSwingIndex (Function)

Disclaimer

The `AccumSwingIndex` series function calculates the accumulation swing index based on a running total of the `SwingIndex` function value.

Syntax

```
AccumSwingIndex
```

Returns (Double)

A positive or negative number between -100 and 100 for the current bar.

Parameters

None.

Remarks

The `SwingIndex` function was developed to help cut through the maze of `Open`, `High`, `Low` and `Close` prices to indicate the real strength and direction of the market. The `SwingIndex` function looks at the `Open`, `High`, `Low`, and `Close` values for a two-bar period.

The Swing Index uses `DailyLimit` as a weighting factor to set the upper and lower limits of the calculation. Since `DailyLimit` is the most a futures price may fluctuate in a day, this calculation generally applies only to futures securities.

This is a cumulative calculation type function, meaning that the current bar's value for this function is based on the values of all previous bars.

`AccumSwingIndex` is normally applied to a futures symbol.

Example

Assigns the `AccumSwingIndex` value to `Value1`, then plots `Value1`:

```
Value1 = AccumSwingIndex;

Plot1(Value1, "AccSwng");
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

AdaptiveMovAvg (Function)

Disclaimer

The AdaptiveMovAvg series function calculates an adaptive moving average based on a variable speed exponential moving average. The number of bars used in the average is reduced as the price action becomes steadier with lower volatility. The number of bars used increases when the price action becomes more volatile.

This function is used to smooth the values of other functions or values. It calculates an efficiency ratio based on the trending tendency of prices, then a smoothing factor is calculated and applied to the calculation.

Syntax

```
AdaptiveMovAvg(Price,EffRatioLength,FastAvgLength,SlowAvgLength)
```

Returns (Double)

A numeric value containing a variable speed exponential moving average..

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to consider.
EffRatioLength	Numeric	Sets he number of bars used to calculate an efficiency ratio.
FastAvgLength	Numeric	Sets the number of bars used to calculate a fast smoothing factor.
SlowAvgLength	Numeric	Sets the number of bars used to calculate a slow smoothing factor.

Remarks

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

The value for the Length input parameter should always be a whole number greater than 0.

Example

Assigns the AdaptiveMovAvg value of the bar Close to Value1, then plots Value1:

```
Value1 = AdaptiveMovAvg(Close, 10, 2, 30);
Plot1(Value1, "ADPMA");
```

Assigns the AdaptiveMovAvg value for the RSI function to Value1, then plots Value1:

```
Value1 = AdaptiveMovAvg(RSI(Close,14), 10, 2, 30);
Plot1(Value1, "ADPRSI");
```

AdvanceDeclineDiff (Function)

Disclaimer

The AdvanceDeclineDiff series function calculates the cumulative difference between advancing versus declining data sets; both data sets must be present in the same chart.

Syntax

```
AdvanceDeclineDiff(AdvIssues,DecIssues)
```

Returns (Integer)

A positive or negative number for the current bar.

Parameters

Name	Type	Description
AdvIssues	Numeric	Specifies a the advancing issues price data series. Normally the Close of Data1 referencing the close of an advancing issues symbol.
DecIssues	Numeric	Specifies a the declining issues price data series. Normally the Close of Data2 referencing the close of a declining symbol.

Remarks

Traditionally, this calculation has been used with NYSE advancing issues and declining issues data to provide a gauge of market activity and volatility without regard to price direction, but both the NASDAQ and AMEX stock exchanges also provide this information.

Difference = x - y

You will need to create a multidata chart with both advancing issues and declining issues in the chart to use this function.

This is a cumulative calculation type function, meaning that the current bar’s value for this function is based on the values of all previous bars.

Advancing issues and Declining issues symbols for the three major stock exchanges can be found using the symbol lookup feature. For more information, see Using the Symbol Lookup Feature.

The TradeStation Data Network symbols for NYSE advancing and declining issues are $ADV and $DECL.

Example

Assigns the cumulative difference between the Close of Data1 (Advancing Issues symbol) and the Close of Data2 (Declining Issues symbol) to Value1; then plots Value1:

```
Value1 = AdvanceDeclineDiff(Close of Data1, Close of Data2);

Plot1(Value1, "AdvDecDiff");
```

AdvanceDeclineRatio (Function)

Disclaimer

The AdvanceDeclineRatio function calculates the ratio between advancing versus declining data sets, both data sets must be present in the same chart.

Syntax

```
AdvanceDeclineRatio(AdvIssues,DecIssues)
```

Returns (Double)

A positive or negative number for the current bar.

Parameters

Name	Type	Description
AdvIssues	Numeric	Specifies a the advancing issues price data series. Normally the Close of Data1 referencing the close of an advancing issues symbol.
DecIssues	Numeric	Specifies a the declining issues price data series. Normally the Close of Data2 referencing the close of a declining symbol.

Remarks

Traditionally, this calculation has been used with NYSE advancing issues and declining issues data to provide a gauge of market activity and volatility without regard to price direction, but both the NASDAQ and AMEX stock exchanges also provide this information.

Ratio = x / y

You will need to create a multidata chart with both advancing issues and declining issues in the chart to use this function.

Advancing issues and Declining issues symbols for the three major stock exchanges can be found using the symbol lookup feature. For more information, see Using the Symbol Lookup Feature.

The TradeStation Data Network symbols for NYSE advancing and declining issues are $ADV and $DECL.

Example

Assigns the cumulative ratio between the `Close of Data1` (Advancing Issues symbol) and the `Close of Data2` (Declining issues symbol) to `Value1`; then plots `Value1`:

```
Value1 = AdvanceDeclineRatio (Close of Data1, Close of Data2);
Plot1(Value1, "AdvDecR");
```

ADX (Function)

Disclaimer

The ADX series function returns the average directional movement index (DMI) for a security.

Syntax

```
ADX(Length)
```

Returns (Double)

A positive numeric value for the current bar.

Parameters

Name	Type	Description
Length	Numeric	Sets the number of bars to consider for the average directional movement index calculation.

Remarks

ADX attempts to measure the trending quality of a security independent of direction. The greater the ADX value, the stronger a security is trending.

The DirMovement function calculates the DMI and ADX values.

The value for the Length input parameter should always be a whole number.

Example

Assigns the ADX value to Value1, where the ADX length is 14 bars, then plots Value1:

```
Value1 = ADX(14);
Plot1(Value1, "ADX");
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

See Also

ADXCustom, DirMovement

ADXCustom (Function)

Disclaimer

The ADXCustom series function returns the average of the directional movement index (DMI) for a security.

Syntax

```
ADXCustom(PriceH,PriceL,PriceC,Length)
```

Returns (Double)

A positive numeric value for the current bar.

Parameters

Name	Type	Description
PriceH	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement high price.
PriceL	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement low price.
PriceC	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement close price.
Length	Numeric	Sets the number of bars to consider for the directional movement index plus calculation.

Remarks

ADXCustom attempts to measure the trending quality of a security independent of direction. The greater the ADXCustom value, the stronger a security is trending.

ADXCustom provides added flexibility over the normal ADX function by allowing you to specify what High, Low, and Close values to use. This is generally used to reference multi-data elements other than Data1.

The DirMovement function calculates the DMI and ADX values.

The value for the Length input parameter should always be a whole number.

Example

Assigns the ADXCustom value of the second data element in a chart (Data2) to Value1, where the ADX length is 14 bars, then plots Value1:

```
Value1 = ADXCustom(High of Data2, Low of Data2, Close of Data2,14);
Plot1(Value1, "ADXCus");
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

See Also

ADX, DirMovement

ADXR (Function)

Disclaimer

The ADXR series function returns the ADX rating value which is the (ADX + ADX of some bars ago / 2).

Syntax

```
ADXR(Length)
```

Returns (Double)

A positive numeric value for the current bar.

Parameters

Name	Type	Description
Length	Numeric	Sets the number of bars to consider for the average directional movement index calculation.

Remarks

ADXR attempts to rate securities on a rating scale that is representative of the directional movement.

The DirMovement function calculates the DMI, ADXR, and ADX values.

The value for the Length input parameter should always be a whole number.

Example

Assigns the ADXR value to Value1, where the ADXR length is 14 bars, then plots Value1:

```
Value1 = ADXR(14);
Plot1(Value1, "ADXR");
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

See Also

ADXRCustom, DirMovement

ADXRCustom (Function)

Disclaimer

The ADXRCustom series function returns the custom ADX rating value which is the (ADX + ADX of some bars ago / 2).

Syntax

```
ADXRCustom(PriceH,PriceL,PriceC,Length)
```

Returns (Double)

A positive numeric value for the current bar.

Parameters

Name	Type	Description
PriceH	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement high price.
PriceL	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement low price.
PriceC	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement close price.
Length	Numeric	Sets the number of bars to consider for the directional movement index plus calculation.

Remarks

ADXRCustom attempts to rate securities on a rating scale that is representative of directional movement.

ADXRCustom provides added flexibility over the normal ADXR function by allowing you to specify what High, Low, and Close values to use. This is generally used to reference multi-data elements other than Data1.

The DirMovement function calculates the DMI, ADRX, and ADX values.

The value for the Length input parameter should always be a whole number greater than 0.

Example

Assigns the ADXRCustom value of the second data element in a chart (Data2) to Value1, where the ADXR length is 14 bars, then plots Value1:

```
Value1 = ADXRCustom(High of Data2, Low of Data2, Close of Data2,14);
Plot1(Value1, "ADXRCus");
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

See Also

ADXR, DirMovement

ArmsIndex (Function)

Disclaimer

The ArmsIndex function calculates the ratio between volume weighted advancing and declining issues.

Syntax

```
ArmsIndex(AdvIssues,AdvVolume,DecIssues,DecVolume)
```

Returns (Double)

A numeric value of the arms index for the current bar.

Parameters

Name	Type	Description
AdvIssues	Numeric	Specifies an advancing issues price data series. Normally the Close of Data1 referencing the close of an advancing issues symbol.
AdvVolume	Numeric	Specifies an advancing volume data series. Normally the Volume of Data1 referencing the volume of an advancing issues symbol.
DecIssues	Numeric	Specifies an declining issues price data series. Normally the Close of Data2 referencing the close of a declining issues symbol.
DecVolume	Numeric	Specifies an declining volume data series. Normally the Volume of Data2 referencing the volume of a declining issues symbol.

Remarks

The Arms Index was developed by Richard Arms; it can also be referred to as the TRIN. It is commonly used as a short term trading tool that attempts to show whether volume is flowing into advancing or declining stocks. The resulting index may also be smoothed by an average.

You will need to create a multidata chart with advancing issues and declining issues both in the chart to use this function.

Advancing issues and Declining issues symbols for the three major stock exchanges can be found using the symbol lookup feature. For more information, see Using the Symbol Lookup Feature.

The TradeStation data network symbols for NYSE advancing and declining issues are $ADV and $DECL.

Example

Assigns the 21 bar moving average of the ArmsIndex to Value1, then plots Value1:

```
Value1 = Average(ArmsIndex(Close of Data1, Close of Data2, Volume of Data1, Volume
of Data2), 21);
Plot1(Value1, "AvgARMS");
```

At Sign (@) (Identifier)

Disclaimer

The At sign (`@`) is used to identify a user-created function. Using the (`@`) symbol has no affect on the logic or calculation of the function.

Syntax

```
@myfunction
```

Returns (None)

Has no return value.

Parameters

None

Remarks

The `@` identifier also forces the function to be called over the declared variable of the same name.

Example

Identifies the user function `myAverage` as a user-created function.

```
Value1 = @myAverage(Close, 10);
```

Average (Function)

Disclaimer

The Average function calculates the standard arithmetic mean of prices or values over a range of bars. It may also be called a moving average, since the values are recalculated for every bar.

Syntax

```
Average(Price,Length)
```

Returns (Double)

A numeric value for the current bar.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Length	Numeric	Sets the number of bars to consider.

Remarks

The Average function is often used to smooth the values of functions or indicators.

Average (arithmetic mean) is the sum of *n* numbers divided by *n.*

Average = (1+3+5) / 3 = 3

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as (High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

The value for the Length input parameter should always be a positive whole number.

Examples

Assigns the 9 bar moving average of the Close to Value1, then plots Value1:

```
Value1 = Average(Close,9);
Plot1(Value1, "AvgClose");
```

Assigns the 15 bar moving average of the range;(High + Low) /2 to Value1, then plots Value1:

```
Value1 = Average((High + Low)/2,15);
Plot1(Value1, "AvgRng");
```

Assigns the 10 bar moving average of the RSI function to Value1, then plots Value1:

```
Value1 = Average(RSI(Close,14),10);
Plot1(Value1, "AvgRSI");
```

Assigns the 20 bar moving average of the custom variable myRange to Value1, then plots Value1:

```
Vars: myRange(0);
MyRange = ((High[1]-Low[1]) + (High - Low))/2;
Value1 = Average(myRange, 20);
Plot1(Value1, "AvgRng");
```

AverageArray (Function)

Disclaimer

The AverageArray function calculates a standard Average of all the values within an array.

Syntax

```
AverageArray(PriceArray,Size)
```

Returns (Double)

The average of the values in a specified array..

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing values upon which the average is calculated.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.

Remarks

The average is the sum of the numbers in an array divided by the number of array elements, such as (1+2+3+4+5) / 5 = 3

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The AverageArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Assigns to Value1 the 10 bar moving average of the specified array.

```
Array: myArray[10](0);
{… (assign values to array) }
Value1 = AverageArray(myArray, 10);
```

AverageFC (Function)

Disclaimer

The AverageFC (Fast Calculation) series function is an Average of prices or values for some number of bars. It may also called a moving average, since the values are recalculated for every bar.

Syntax

```
AverageFC(Price, Length)
```

Returns (Double)

A numeric value for the current bar.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Length	Numeric	Sets the number of bars to consider.

Remarks

The AverageFC function is often used to smooth the values of functions or indicators.

AverageFC (fast calculation arithmetic mean) subtracts the oldest value in the list of elements, adds the newest value, and then divides by the number of elements.

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as (High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

The value for the Length input parameter should always be a whole number, greater than 0.

Note This series function returns exactly the same values as Average except that it uses a fast calculation method that takes slightly more memory than the non-FC version.

Examples

Assigns the 9 bar fast calculation moving average of the Close to Value1, then plots Value1:

```
Value1 = AverageFC(Close,9);
Plot1(Value1, "AvgClose");
```

Assigns the 15 bar fast calculation moving average of the range; (High + Low)/2 to Value1, then plots Value1:

```
Value1 = AverageFC((High + Low)/2,15);
Plot1(Value1, "AvgRng");
```

Assigns the 10 bar fast calculation moving average of the RSI function to Value1, then plots Value1:

```
Value1 = AverageFC(RSI(Close,14),10);
Plot1(Value1, "AvgRSI");
```

Assigns the 20 bar fast calculation moving average of the custom variable myRange to Value1, then plots Value1:

```
Vars: myRange(((High[1]-Low[1]) + (High - Low))/2;
Value1 = AverageFC(myRange, 20);
Plot1(Value1, "AvgRng");
```

AvgDeviation (Function)

Disclaimer

The AvgDeviation function is an Average of the absolute value of the difference of each data point from the mean and dividing the sum by the number of elements.

Syntax

```
AvgDeviation(Price,Length)
```

Returns (Double)

A positive numeric of the average deviation for the current bar.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to be considered.
Length	Numeric	Sets the number of bars to be considered.

Remarks

AvgDeviation is a measure of the variability in a data set.

The AvgDeviation function first calculates a standard average (mean) of the Price input parameter using the Length input parameter for the number of trailing bars. It then measures the absolute difference (ABSValue, which removes the negative sign) of the Price input parameter from the mean for each data point and then averages the sum of the differences.

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

The value for the Length input parameter should always be a whole number greater than 0.

Example

Assigns the 9 bar average deviation of the Close from the mean to Value1, then plots Value1:

```
Value1 = AvgDeviation (Close,9);
Plot1(Value1, "AvgDev");
```

AvgDeviationArray (Function)

Disclaimer

The AvgDeviationArray function calculates the average difference between the array values and the array average.

Syntax

```
AvgDeviationArray(PriceArray,Size)
```

Returns (Double)

A numeric value for a specified array, for the current bar. If no average calculation is performed as a result of improper array sizing or reference, the function will return -1.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing values upon which the average deviation is calculated.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.

Remarks

AvgDeviation is a measure of the variability in a data set.

The AvgDeviationArray function first calculates a standard average (mean) of the specified PriceArray using the Length input parameter for the number of trailing bars. Next it measures the absolute difference (ABSValue, which removes the negative sign) of each array data element from the mean and averages the sum of the differences.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The AvgDeviationArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Assigns to Value1 the 10 bar average deviation from the mean for a specified array.

```
Array: myArray[10](0);
{… (assign values to array) }
Value1 = AvgDeviationArray (myArray, 10);
```

AvgPrice (Function)

Disclaimer

The AvgPrice function calculates the average price of a bar, by averaging the four bar points: High, Low, Open, and Close.

Syntax

```
AvgPrice
```

Returns (Double)

A positive numeric value for the current bar.

Parameters

None

Remarks

The AvgPrice function is provided as a convenient way to integrate the average bar price into your analysis.

AvgPrice offers a smoothing effect of extreme price points on a bar.

AvgPrice = (Open + High + Low + Close)/4, or (High + Low + Close)/3, (if Open is not applicable).

Examples

Assigns the average price on a bar to Value1, then plots Value1:

```
Value1 = AvgPrice;
Plot1(Value1, "AvgPrc");
```

Assigns the 10 bar average of the average price on a bar to Value1, then plots Value1:

```
Value1 = Average(AvgPrice, 10);
Plot1(Value1, "AvgAvgPrc");
```

AvgTrueRange (Function)

Disclaimer

The AvgTrueRange function calculates the average of the TrueRange for some number of bars.

Syntax

```
AvgTrueRange(Length)
```

Returns (Double)

A numeric value containing the average TrueRange over some number of bars.

Parameters

Name	Type	Description
Length	Numeric	Sets the number of bars to consider for the average true range.

Remarks

AvgTrueRange is used to smooth out price bars with volatility that is higher or lower than normal.

TrueRange is defined as the larger of the following:

- The distance between today’s High and today's Low.
- The distance between today’s High and yesterday's Close.
- The distance between today’s Low and yesterday's Close.

The value for the Length input parameter should always be a positive whole number greater than 0.

Example

Assigns to Value1 the 10 bar average of the TrueRange for each bar, then plots the result:

```
Value1 = AvgTrueRange(10);
Plot1(Value1, "AvgTRng");
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

BarAnnualization (Function)

Disclaimer

The `BarAnnualization` function returns an annualization factor based on the bar interval (daily, weekly, or monthly) of the chart.

Syntax

```
BarAnnualization
```

Returns (Double)

A positive numeric value for the current bar.

Parameters

None

Remarks

This function does not work with intraday charts.

You can use the `BarAnnualization` factor to calculate and compare the annualized standard deviation of some values over multiple time frames; it is also used in options analysis and other annualized calculations.

- Daily annualization factor = 19.1049 (square root of 365, using calendar days not trading days)
- Weekly annualization factor = 7.2111 (square root of 52)
- Monthly annualization factor = 3.4641 (square root of 12)

Example

Assigns the annualized Standard Deviation of the `Close` over 10 bars on a daily chart to `Value1`, and assigns the annualized Standard Deviation of the `Close` over 10 bars on a weekly chart to `Value2`, in a multi-data chart; then plots `Value1` and `Value2`:

```
Value1 = StdDev(Close, 10) * BarAnnualization;
Value2 = StdDev(Close of data2, 10) * BarAnnualization of data2;
Plot1(Value1,"DStdDev");
Plot2(Value2,"WStdDev");
```

BarNumber (Function)

Disclaimer

The BarNumber series function assigns a reference number to each bar after MaxBarsBack is satisfied.

Syntax

```
BarNumber
```

Returns (Integer)

A positive numeric reference value for each bar on the chart.

Parameters

None

Remarks

MaxBarsBack is the minimum number of referenced historical bars required at the beginning of a chart to begin calculating trading strategies, analysis techniques, and functions. For example, a 10-bar moving average would require a MaxBarsBack setting of 9 to calculate, which is 9 historical bars and the current bar.

Since BarNumber is based on MaxBarsBack, if there are 500 bars in a chart, with a MaxBarsBack setting of 50, the next bar after the 50th bar on the chart moving left to right, will be BarNumber = 1. The last bar on the chart (most recent) will be BarNumber = 451.

BarNumber is often used to identify a particular bar or number of bars because of some special occurrence or situation that you want to test or factor into your analysis.

The BarNumber function is similar to the reserved word CurrentBar. However, CurrentBar does not allow previous bar references: BarNumber[5] (of five bars ago) is correct, however, Currentbar[5] is incorrect and does not work.

Examples

Assigns the BarNumber for each bar to Value1, then plots Value1:

```
Value1 = BarNumber;

Plot1(Value1, "BarNum");
```

Assigns the BarNumber to Value1 when Condition1 is true, and assigns the number of bars since Condition1 occurred to Value2:

```
if Condition1 then
 Value1 = BarNumber;
if BarNumber > Value1 then
 Value2 = CurrentBar - Value1;
```

BearishDivergence (Function)

Disclaimer

The BearishDivergence function identifies higher occurrences of Pivot highs in one value, accompanied by lower Pivot highs in another value. A Pivot High is a significant high value preceded and proceeded by some number of lower values.

BearishDivergence is normally identified when a price (High) makes a Pivot high that is higher than the previous Pivot high, and an oscillator (Stochastic, RSI, BollingerBands) makes a Pivot high that is lower than the previous Pivot high.

Syntax

```
BearishDivergence(Price1,Price2,Strength,Length)
```

Returns (Integer)

The function returns a value of 1 if the BearishDivergence condition is True for the current bar, otherwise it returns 0.

Parameters

Name	Type	Description
Price1	Numeric	Specifies which bar value (price, function, or formula) to use for the first pivot point.
Price2	Numeric	Specifies which bar value (price, function, or formula) to use for the second pivot point.
Strength	Numeric	Sets the number of bars on either side of the pivot high point.
Length	Numeric	Sets the maximum number of bars between pivot high points.

Remarks

The input parameters Price1 and Price2 can be a bar value such as Close, High, Low, Open, or Volume. They can also be any mathematical calculation such as High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

Increasing the Strength input parameter will normally decrease the number of True occurrences.

Increasing the Length input parameter will normally increase the chances for True of occurrences.

The value for Length and Strength should always be a positive whole number greater than 0.

Example

Plots a ShowMe on the High when a BearishDivergence occurs between a Pivot High and the Pivot high of the RSI oscillator on any bar, where the Pivot Strength is 2 and the Pivot Length is 20:

```
if BearishDivergence(High, RSI(Close,14), 2, 20) = 1 then
 Plot1(High, "BearDrvg");
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

See Also

BullishDivergence

BjerkStensCall (Function)

Disclaimer

The BjerkStensCall function is used to approximate the theoretical value of American call options; and through a put-call transformation, the function can also be used to price American put options.

Syntax

```
BjerkStensCall(AssetPr, StrikePr, YearsLeft, Rate, Carry, Volty);
```

Returns (Double)

A numeric value representing the theoretical value of an American Call or Put option using the Bjerksund & Stensland model.

Parameters

Name	Type	Description
AssetPr	Numeric	Specifies the price of the underlying asset.
StrikePr	Numeric	Specifies the strike price of the option.
YearsLeft	Numeric	Sets the amount of time left until option expiration in terms of years (enter 1 to stand for 365 days).
Rate	Numeric	Sets the short-term risk free interest rate, usually the 90-day T-Bill, (enter .049 for 4.9%).
Carry	Numeric	Sets the cost of carrying the underlying asset as a percentage in decimal points, usually the short-term risk free interest rate (enter .049 for 4.9%).
Volty	Numeric	Sets the volatility of the underlying asset as a percentage in decimal points (enter .225 for 22.5%).

Remarks

The value for Carry differs based on the type of asset:

- Non-dividend paying stock - Set to the current annual risk-free interest rate.
- Dividend paying stock - Set the current annual risk-free interest rate minus the annual dividend yield percentage. Example: if the current risk-free interest rate is 4.5% and the stock's annual dividend yield is 2.4%, then the cost of carry could be approximated as 4.5% - 2.4% = 2.1%.
- Futures contract - Set to 0 (zero).
- Currency contract - Set to the current "domestic" risk-free interest rate minus current "foreign" risk-free interest rate. Note: the domestic and foreign interest rates depend on how the currency cross-rate is quoted.

The value for Volty can also be a reserved word value such as IVolatility.

Usage

If the BjerkStensCall function is being called to obtain the model price of a Put option, then the value passed as the input Rate should be Rate-Carry, and the value passed to the input Carry should be -Carry (based on the put-call transformation). Although written for calls, this function can be used for pricing American puts via a put-call transformation, which requires that the arguments be passed in as follows:

	INPUTS FOR CALLS	INPUTS FOR PUTS
Input1	AssetPr	StrikePr
Input2	StrikePr	AssetPr
Input3	YearsLeft	YearsLeft
Input4	Rate	Rate-Carry
Input5	Carry	-Carry
Input6	Volty	Volty

Note BjerkStensCall is used by the OptionsComplex function to calculate the price for American puts by using the above transformation.

Example

Assigns to `Value1` the theoretical price of an American non-paying dividend Call option with a 6 month (1/2 year) expiration from the current date with the short-term 90-day T-Bill at 4.9%.

```
  Value1 = BjerkStensCall(36, 34, .5, 0, 4.9, IVolatility) ;
```

For additional information and details, you may wish to consult the following reference texts:

Chriss, Neil A. *Black-Scholes and Beyond: Option Pricing Models*. McGraw-Hill, 1997.
Haug, Espen Gaarder. *Option Pricing Formulas*. McGraw-Hill, 1998.

BjerkStensPhi (Function)

Disclaimer

The BjerkStensPhi function is used by the BjerkStensCall function to calculate the theoretical value of American Call and Put options.

Syntax

```
BjerkStensPhi(AssetPr, YearsLeft, MyGamma, MyH, TriggerPr, Rate, Carry, Volty);
```

Returns (Double)

A numeric value representing the theoretical value of an American Call or Put option using the Bjerksund & Stensland model.

Parameters

Name	Type	Description
AssetPr	Numeric	Specifies the price of the underlying asset.
YearsLeft	Numeric	Sets the amount of time left until option expiration in terms of years (enter 1 to stand for 365 days).
MyGamma	Numeric	Sets the Gamma to consider for the calculation.
MyH	Numeric	Sets the StrikePr or TriggerPr as determined by the calling function BjerkStensCall.
TriggerPr	Numeric	Sets the flat boundary of exercise strategy, as detemined by the calling function BjerkStensCall.
Rate	Numeric	Sets the short-term risk free interest rate, usually the 90-day T-Bill, (enter .049 for 4.9%).
Carry	Numeric	Sets the cost of carrying the underlying asset as a percentage in decimal points, usually the short-term risk free interest rate (enter .049 for 4.9%).
Volty	Numeric	Sets the volatility of the underlying asset as a percentage in decimal points (enter .225 for 22.5%).

Remarks

The value for Carry differs based on the type of asset:

- Non-dividend paying stock - Set to the current annual risk-free interest rate.
- Dividend paying stock - Set the current annual risk-free interest rate minus the annual dividend yield percentage. Example: if the current risk-free interest rate is 4.5% and the stock's annual dividend yield is 2.4%, then the cost of carry could be approximated as 4.5% - 2.4% = 2.1%.
- Futures contract - Set to 0 (zero).
- Currency contract - Set to the current "domestic" risk-free interest rate minus current "foreign" risk-free interest rate. Note: the domestic and foreign interest rates depend on how the currency cross-rate is quoted.

The value for Volty can also be a reserved word value such as IVolatility.

Note BjerkStensPhi is used by the BjerkStensCall and OptionsComplex functions.

For additional information and details, you may wish to consult the following reference texts:

Chriss, Neil A. *Black-Scholes and Beyond: Option Pricing Models*. McGraw-Hill, 1997.
Haug, Espen Gaarder. *Option Pricing Formulas*. McGraw-Hill, 1998.

BlackModel (Function)

Disclaimer

The BlackModel function calculates the theoretical value of an option based on the Black option pricing model. BlackModel is designed to be used for European future options.

Syntax

```
BlackModel(DaysLeft, StrikePr, AssetPr, Rate100, Volty100, PutCall);
```

Returns (Double)

A numeric value representing the theoretical value of a European future option.

Parameters

Name	Type	Description
DaysLeft	Numeric	Sets the number of calendar days left until expiration of the option.
StrikePr	Numeric	Specifies the strike price of the option.
AssetPr	Numeric	Specifies the price of the underlying asset.
Rate100	Numeric	Sets the short-term risk free interest rate, usually the 90 day T-Bill, entered as a percent (enter 4.9 for 4.9%).
Volty100	Numeric	Sets the volatility of the underlying asset as a percent (enter 22.5 for 22.5%).
PutCall	Numeric	Specify if it is a Put or Call option. Put or 2 = Puts; Call or 3 = Calls.

Remarks

The input parameter DaysLeft can also be a numeric function such as DaystoExpiration or Next3rdFriday.
The input parameter Volty100 can also use a reserved word value such as IVolatility *100.

Example

Assigns to Value1 the theoretical price of a European future call option using the DaystoExpiration function to calculate the number of days left in the option until it expires in January of 2007 from today with the short-term 90-day T-Bill at 4.9%.

```
Value1 = BlackModel(DaystoExpiration(1, 107), Strike, Close, 4.9, 22.5, Call);
```

See Also

BlackScholes, OS_Binomial.

BlackScholes (Function)

Disclaimer

The BlackScholes function calculates the theoretical value of an option based on the Black-Scholes option pricing model.   BlackScholes is designed to be used for European non-dividend paying stock options.

Syntax

```
BlackScholes(DaysLeft, StrikePr, AssetPr, Rate100, Volty100, PutCall);
```

Returns (Double)

A numeric value representing the theoretical value of a European non-dividend paying stock option.

Parameters

Name	Type	Description
DaysLeft	Numeric	Sets the number of calendar days left until expiration of option.
StrikePr	Numeric	Sets the strike price of the option.
AssetPr	Numeric	Sets the price of underlying asset.
Rate100	Numeric	Sets the short-term risk free interest rate, usually the 90-day T-Bill, as a percentage (enter 4.9 for 4.9%).
Volty100	Numeric	Sets the volatility of the underlying asset as a percentage (enter 22.5 for 22.5%).
PutCall	Numeric	Specify if it is a Put or Call option. Put or 2 = Puts; Call or 3 = Calls.

Remarks

The input parameter DaysLeft can also be a numeric function such as DaystoExpiration or Next3rdFriday.
The input parameter Volty100 can also use a reserved word value such as IVolatility *100.

Example

Assigns to Value1 the theoretical price of a European non-paying dividend call option using the DaystoExpiration function to calculate the number of days left in the option until it expires in January of 2007 from today with the short-term 90-day T-Bill at 4.9%.

```
Value1 = BlackScholes(DaystoExpiration(1, 107), Strike, Close, 4.9, 22.5, Call);
```

See Also

BlackModel, OS_Binomial.

BollingerBand (Function)

Disclaimer

The BollingerBand function calculates an *n* standard deviation (StdDev) line (usually 2 StdDevs) above or below a center-line simple moving average.

Syntax

```
BollingerBand(Price, Length, NumDevs)
```

Returns (Double)

A numeric value for the current bar.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to be considered for the center-line average.
Length	Numeric	Sets the number of bars to consider for the center-line average.
NumDevs	Numeric	Sets the number of standard deviations above (positive) or below (negative) the center-line average.

Remarks

Normally BollingerBands are used with price data, but they can also be used with indicators and other calculated values.

The BollingerBand function can be interpreted in many ways and can be used in multiple time frames. The traditional interpretation would look for a bar to cross over one on the bands and then cross back over, signaling a potential market reversal.

The value for the Length input parameter should always be a positive whole number greater than 0.

The number of Standard Deviations represents the percentage of values that lie within the normal distribution range of values. At 2 standard deviations over 95 percent of all values lie within the normal distribution of values. At 3 standard deviations over 99 percent of all values lie within the normal distribution of values.

By using a number of standard deviations within the normal distribution range of values, the BollingerBand adjusts for price volatility.

The input parameter NumDevs can be any decimal value ranging from -3 to +3. (-3, -2.5, -1.5, 1.5, 2.5, 3).

Example

Assigns to Value1 the upper BollingerBand, and assigns to Value2 and plots the lower BollingerBand, for each bar based on 2 standard deviations a simple 20 bar average of the Close, then plots Value1 and Value2:

```
   Value1 = BollingerBand(Close,20,2);
   Value2 = BollingerBand(Close,20,-2);
   Plot1(Value1, "UpperBB");
   Plot2(Value2, "LowerBB");
```

Assigns to Value1 the upper BollingerBand, and assigns to Value2 and plots the lower BollingerBand, for each bar based on 2 standard deviations of a simple 5 bar average of the RSI, then plots Value1, Value2, and the RSI value:

```
   Value1 = BollingerBand(RSI (Close,14),5,2);
   Value2 = BollingerBand(RSI (Close,14),5,-2);
   Plot1(Value1, "UpperBB");
   Plot2(Value2, "LowerBB");
   Plot3(RSI(Close,14, "RSI");
```

Reference

John Bollinger, CFA, President, Bollinger Capital Management, Inc. P.O. Box 3358, Manhattan Beach, CA 90266.

BullishDivergence (Function)

Disclaimer

The BullishDivergence function identifies lower occurrences of Pivot lows in one value, accompanied by higher Pivot lows in another value. A Pivot Low is a significant low value preceded and proceeded by some number of higher values.

BullishDivergence is normally identified when a price (Low) makes a Pivot low that is lower than the previous Pivot low, and an oscillator (Stochastic, RSI, Bollinger Bands) makes a Pivot low that is higher than the previous Pivot low.

Syntax

```
BullishDivergence(Price1,Price2,Strength,Length)
```

Returns (Integer)

The function returns a value of 1 if the BullishDivergence condition is True for the current bar, otherwise it returns 0.

Parameters

Name	Type	Description
Price1	Numeric	Specifies which bar value (price, function, or formula) to use for the first pivot point.
Price2	Numeric	Specifies which bar value (price, function, or formula) to use for the second pivot point.
Strength	Numeric	Sets the number of bars on either side of the pivot high point.
Length	Numeric	Sets the maximum number of bars between pivot high points.

Remarks

The input parameters Price1 and Price2 can be a bar value such as Close, High, Low, Open, or Volume. They can also be any mathematical calculation such as High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

Increasing the Strength input parameter will normally decrease the number of True occurrences.

Increasing the Length input parameter will normally increase the chances for True of occurrences.

The value for Length and Strength should always be a positive whole number greater than 0.

Example

Plots a ShowMe on the Low when a BullishDivergence occurs between a Pivot Low and the Pivot low of the RSI oscillator on any bar, where the Pivot Strength is 2 and the Pivot Length is 20:

```
if BullishDivergence(Low, RSI(Close,14), 2, 20) = 1 then
 Plot1(Low, "BullDrvg");
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

See Also

BearishDivergence

C_3WhSolds_3BlkCrows (Function)

Disclaimer

The C_3WhSolds_3BlkCrows function identifies occurrences of two Japanese candlestick patterns; Three Advancing White Soldiers and Three Black Crows.

Three Advancing White Soldiers is defined as:

- Three advancing consecutive green candles
- The three lines should close at, or near, their highs.
- Opening of the second and third candle should be inside the body of the previous candle.

Three Black Crows is defined as:

- Three declining consecutive red candles
- The three lines should close at, or near, their lows.
- Opening of the second and third candle should be inside the body of the previous candle.

Syntax

```
C_3WhSolds_3BlkCrows(Length, Percent, o3WhiteSoldiers, o3BlackCrows)
```

Returns (Integer)

Each pattern is represented by an output parameter (o3WhiteSoldiers or o3BlackCrows) that returns a positive one (1) when the matching candlestick pattern is found on the current bar or returns a zero (0) if the pattern is not found. The C_3WhSolds_3BlkCrows function itself returns a value of 1.

Procedures

Name	Type	Description
Length	Numeric	Length used to calculate the average body.
Percent	Numeric	Threshold proximity percent of the close to the high or low.
o3WhiteSoldiers	Numeric	Output variable returns a 1 if the Three Advancing White Soldiers candlestick pattern exists on the current bar, or 0 if the pattern does not exist.
o3BlackCrows	Numeric	Output variable returns a 1 if the Three Black Crows candlestick pattern exists on the current bar, or 0 if the pattern does not exist.

Example

Plots a ShowMe on the Low of the current bar when a Three Advancing White Soldiers candlestick pattern is found on the current bar based on a Length of 14 and a Percent of 5:

```
Value1 = C_3WhSolds_3BlkCrows(14, 5, o3WhiteSoldiers, o3BlackCrows)
if o3WhiteSoldiers = 1 then
 Plot1(Low, "3WhiteSoldiers");
```

C_BullEng_BearEng (Function)

Disclaimer

The C_BullEng_BearEng function identifies occurrences of two Japanese candlestick patterns; Bullish Engulfing and Bearish Engulfing.

Bullish Engulfing is defined as:

- The body of a green candle is larger and engulfs both the open and close of a red candle on the previous bar
- The price trend is declining
- The body of the green candle is greater than the average body

Bearish Engulfing is defined as:

- The body of a red candle is larger and engulfs both the open and close of a green candle on the previous bar
- The price trend is advancing
- The body of the red candle is greater than the average body

Syntax

```
C_BullEng_BearEng(Length, oBullishEngulfing, oBearishEngulfing)
```

Returns (Integer)

Each pattern is represented by an output parameter (oBullishEngulfing or oBearishEngulfing) that returns a positive one (1) when the matching candlestick pattern is found on the current bar or returns a zero (0) if the pattern is not found. The C_BullEng_BearEng function itself returns a value of 1.

Procedures

Name	Type	Description
Length	Numeric	Length used to calculate the average body.
oBullishEngulfing	Numeric	Output variable returns a 1 if the Bullish Engulfing candlestick pattern exists on the current bar, or 0 if the pattern does not exist.
oBearishEngulfing	Numeric	Output variable returns a 1 if the Bearish Engulfing candlestick pattern exists on the current bar, or 0 if the pattern does not exist.

Example

Plots a ShowMe on the Low of the current bar when a Bearish Engulfing candlestick pattern is found on the current bar based on a Length of 14:

```
Value1 = C_BullEng_BearEng(14, oBullishEngulfing, oBearishEngulfing)
if oBearishEngulfing = 1 then
 Plot1(Low, "BearishEngulfing");
```

C_BullHar_BearHar (Function)

Disclaimer

The C_BullHar_BearHar function identifies occurrences of two Japanese candlestick patterns; Bullish Harami and Bearish Harami .

Bullish Harami is defined as:

- The body of a green candle is smaller and inside the body of a red candle on the previous bar
- The price trend is declining
- The body of the previous red candle is greater than the average body

Bearish Harami is defined as:

- The body of a red candle is smaller and inside the body of a green candle on the previous bar
- The price trend is advancing
- The body of the previous green candle is greater than the average body

Syntax

```
C_BullHar_BearHar(Length, oBullishHarami, oBearishHarami)
```

Returns (Integer)

Each pattern is represented by an output parameter (oBullishHarami or oBearishHarami) that returns a positive one (1) when the matching candlestick pattern is found on the current bar or returns a zero (0) if the pattern is not found. The C_BullHar_BearHar function itself returns a value of 1.

Procedures

Name	Type	Description
Length	Numeric	Length used to calculate the average body.
oBullishHarami	Numeric	Output variable returns a 1 if the Bullish Harami candlestick pattern exists on the current bar, or 0 if the pattern does not exist.
oBearishHarami	Numeric	Output variable returns a 1 if the Bearish Harami candlestick pattern exists on the current bar, or 0 if the pattern does not exist.

Example

Plots a ShowMe on the High of the current bar when a Bullish Harami candlestick pattern is found on the current bar based on a Length of 14:

```
Value1 = C_BullHar_BearHar(14, oBullishHarami, oBearishHarami)
if oBullishHarami = 1 then
 Plot1(High, "Bullish Harami");
```

C_Doji (Function)

Disclaimer

The C_Doji function identifies the occurrence of a Doji Japanese candlestick pattern.

Doji is defined as:

- The body of the candle is very small; the close is equal or almost equal to the open by some percent of the range of the bar. Seen as a cross on the chart.

Syntax

```
C_Doji(Percent)
```

Returns (Integer)

The function returns a positive one (1) when the matching candlestick pattern is found on the current bar or returns a zero (0) if the pattern is not found.

Procedures

Name	Type	Description
Percent	Numeric	Doji threshold for the (open - close) as a percent of the range of the bar.

Example

Plots a ShowMe on the Close of the current bar when a Doji candlestick pattern is found on the current bar based on a Percent of 5:

```
if C_Doji(5) = 1 then
 Plot1(Close, "Doji");
```

C_Hammer_HangingMan (Function)

Disclaimer

The C_Hammer_HangingMan function identifies occurrences of two Japanese candlestick patterns; Hammer and Hanging Man.

Hammer is defined as:

- The body of the candle is in the upper half of the bar, and the tail is usually twice as long as the body
- The price trend is declining
- It can be a red or green candle, but not a Doji

Hanging Man is defined as:

- The body of the candle is in the upper half of the bar, and the tail is usually twice as long as the body
- The price trend is advancing
- It can be a red or green candle, but not a Doji

Syntax

```
C_Hammer_HangingMan(Length, Factor, oHammer, oHangingMan)
```

Returns (Integer)

Each pattern is represented by an output parameter (oHammer or oHangingMan) that returns a positive one (1) when the matching candlestick pattern is found on the current bar or returns a zero (0) if the pattern is not found. The C_Hammer_HangingMan function itself returns a value of 1.

Procedures

Name	Type	Description
Length	Numeric	Length used to calculate the average body.
Factor	Numeric	Threshold factor for the size of body in relation to the range of the bar (2 = Tail must be 2x larger than body).
oHammer	Numeric	Output variable returns a 1 if the Hammer candlestick pattern exists on the current bar, or 0 if the pattern does not exist.
oHangingMan	Numeric	Output variable returns a 1 if the Hanging Man candlestick pattern exists on the current bar, or 0 if the pattern does not exist.

Example

Plots a ShowMe on the High of the current bar when a Hammer candlestick pattern is found on the current bar based on a Length of 14 and a Factor of 2:

```
Value1 = C_Hammer_HangingMan(14, 2, oHammer, oHangingMan)
if oHammer = 1 then
 Plot1(High, "Hammer");
```

C_MornDoji_EveDoji (Function)

Disclaimer

The C_MornDoji_EveDoji function identifies occurrences of two Japanese candlestick patterns; Morning Doji Star and Evening Doji Star .

Morning Doji Star s defined a:

- A three candlestick pattern, the first is a long red candle, followed by a Doji candle that gaps lower to form the star, the third is a green candle that closes will into the first candles real body
- The first candle body needs to be larger than the average body

Evening Doji Star is defined as:

- A three candlestick pattern, the first is a long green candle, followed by a Doji candle that gaps lower to form the star, the third is a red candle that closes will into the first candles real body.
- The first candle body needs to be larger than the average body

Syntax

```
C_MornDoji_EveDoji(Length, Percent, oMorningDojiStar, oEveningDojiStar)
```

Returns (Integer)

Each pattern is represented by an output parameter (oMorningDojiStar or oEveningDojiStar) that returns a positive one (1) when the matching candlestick pattern is found on the current bar or returns a zero (0) if the pattern is not found. The C_MornDoji_EveDoji function itself returns a value of 1.

Procedures

Name	Type	Description
Length	Numeric	Length used to calculate the average body.
Percent	Numeric	Doji threshold for the (open - close) as a percent of the range of the bar.
oMorningDojiStar	Numeric	Output variable returns a 1 if the Morning Doji Star candlestick pattern exists on the current bar, or 0 if the pattern does not exist.
oEveningDojiStar	NumericSimple	Returns a 1 if the Evening Doji Star candlestick pattern exists on the current bar, or 0 if the pattern does not exist.

Example

Plots a ShowMe on the Low of the current bar when a Three Advancing White Soldiers candlestick pattern is found on the current bar based on a Length of 14 and a Percent of 5:

```
Value1 = C_3WhSolds_3BlkCrows(14, 5, oMorningDojiStar, oEveningDojiStar)
if oEveningDojiStar = 1 then
 Plot1(Low, "EveningDojiStar");
```

C_MornStar_EveStar (Function)

Disclaimer

The C_MornStar_EveStar function identifies occurrences of two Japanese candlestick patterns; Morning Star and Evening Star.

Morning Star is defined a:

- A three candlestick pattern, the first is a long red candle, followed by a small body candle that gaps lower to form the star, the third is a green candle that closes will into the first candles real body
- The first candle body needs to be larger than the average body

Evening Star is defined as:

- A three candlestick pattern, the first is a long green candle, followed by a small body candle that gaps lower to form the star, the third is a red candle that closes will into the first candles real body.
- The first candle body needs to be larger than the average body

Syntax

```
C_MornStar_EveStar(Length, oMorningStar, oEveningStar)
```

Returns (Integer)

Each pattern is represented by an output parameter (oMorningStar or oEveningStar) that returns a positive one (1) when the matching candlestick pattern is found on the current bar or returns a zero (0) if the pattern is not found. The C_MornStar_EveStar function itself returns a value of 1.

Procedures

Name	Type	Description
Length	Numeric	Length used to calculate the average body.
oMorningStar	Numeric	Output variable returns a 1 if the Morning Star candlestick pattern exists on the current bar, or 0 if the pattern does not exist.
oEveningStar	Numeric	Output variable returns a 1 if the Evening Star candlestick pattern exists on the current bar, or 0 if the pattern does not exist.

Example

Plots a ShowMe on the High of the current bar when a Morning Star candlestick pattern is found on the current bar based on a Length of 14:

```
Value1 = C_MornStar_EveStar(14, oMorningStar, oEveningStar)
if oMorningStar = 1 then
 Plot1(Low, "MorningStar");
```

C_PierceLine_DkCloud (Function)

Disclaimer

The `C_PierceLine_DkCloud` function identifies occurrences of two Japanese candlestick patterns; Piercing Line and Dark Cloud.

Piercing Line is defined as:

- A big green candle which opens below the low of the previous red candle and closes above half the previous bars real body
- The price trend is declining

Dark Cloud is defined as:

- A big red candle which opens above the high of the previous green candle and closes below half the previous bars real body
- The price trend is advancing

Syntax

```
C_PierceLine_DkCloud(Length, oPiercingLine, oDarkCloud)
```

Returns (Integer)

Each pattern is represented by an output parameter (`oPiercingLine` or `oDarkCloud`) that returns a positive one (1) when the matching candlestick pattern is found on the current bar or returns a zero (0) if the pattern is not found. The `C_PierceLine_DkCloud` function itself returns a value of 1.

Procedures

Name	Type	Description
`Length`	Numeric	Length used to calculate the average body.
`oPiercingLine`	Numeric	Output variable returns a 1 if the Piercing Line candlestick pattern exists on the current bar, or 0 if the pattern does not exist.
`oDarkCloud`	Numeric	Output variable returns a 1 if the Dark Cloud candlestick pattern exists on the current bar, or 0 if the pattern does not exist.

Example

Plots a ShowMe on the Low of the current bar when a Dark Cloud candlestick pattern is found on the current bar based on a Length of 14:

```
Value1 = C_PierceLine_DkCloud(14, oPiercingLine, oDarkCloud)
if oDarkCloud = 1 then
 Plot1(Low, "DarkCloud");
```

C_ShootingStar (Function)

Disclaimer

The C ShootingStar function identifies the occurrence of a Shooting Star candlestick pattern

Shooting Star is defined as:

- The body of the candle is small but not a Doji, and near the low of the bar, the tail is usually twice as long as the body
- The price trend is rising
- The body gaps up from the body on the previous bar
- The previous body is green and larger than the average body

Syntax

```
C ShootingStar(Percent)
```

Returns (Integer)

The function returns a positive one (1) when the matching candlestick pattern is found on the current bar or returns a zero (0) if the pattern is not found.

Procedures

Name	Type	Description
`Length`	NumericSimple	Length used to calculate the average body.
`Factor`	NumericSimple	Threshold factor for the size of body in relation to the range of the bar (2 = Tail must be 2x larger than body).

Example

Plots a ShowMe on the Close of the current bar when a ShootStar candlestick pattern is found on the current bar based on a Length of 14 and a Factor of 2:

```
if C ShootingStar(14, 2) = 1 then
 Plot1(Close, "ShootingStar");
```

CalcDate (Function)

Disclaimer

The CalcDate function adds and subtracts days from a reference date. The reference date is a numeric value in the EasyLanguage date format: YYYMMDD (Year Month Day).

For example, 1030101 = Jan. 1st, 2003 or 9906015 = June 15, 1999

Since EasyLanguage and Chart Analysis use this special date format, it is sometimes difficult to add or subtract some number of days from the EasyLanguage date. For example, if you wanted to subtract 15 days from Jan 1, 2003, you might try 1030101 – 15, which would result in an invalid date, CalcDate solves this by returning the correct and valid date of 1021216

Syntax

```
CalcDate(RefDate,DaysChange)
```

Returns (Integer)

A numeric value the represents the date in EasyLanguage format for the current bar.

Parameters

Name	Type	Description
RefDate	Numeric	Specifies a chart date in EasyLanguage date format to use as the reference date for the calculation, entered in YYYMMDD format. (Examples: 1030415, 991015, or 1000505)
DaysChange	Numeric	Sets the days to be added (positive value) or subtracted (negative value).

Remarks

EasyLanguage does not see time and date as unique data formats but sees them and treats them as normal numeric values.

The reserved words Date and CurrentDate return numeric date values in an EasyLanguage format (YYYMMDD).

Example

Assigns to Value1 the calculated date 14 days prior to the current bar date to value1:

```
Value1 = CalcDate(Date,-14);
```

See Also

CalcTime

CalcTime (Function)

Disclaimer

The CalcTime function adds and subtracts minutes from a reference time. The reference time is a numeric value in 24-hour military time format: HHMM (Hour Hour Minute Minute).

For example, 10:15am = 1015, or 1345 = 1:45pm

When working with hour and minutes, it is sometimes difficult to add or subtract some number of minutes with an HHMM time. For example, if you wanted to subtract 15 minutes from noon you might try 1200 – 15, which would result in an answer of 1185, which is not a valid time. CalcTime solves this by returning the correct and valid time of 1145.

Syntax

```
CalcTime(RefTime,MinuteChange)
```

Returns (Integer)

A numeric value that represents the time in 24-hour format for the current bar.

Parameters

Name	Type	Description
RefTime	Numeric	Specifies the chart time in 24-hour time format to use as the reference time for the calculation, entered in HHMM format. (Examples: 0930, 1245, 1600)
MinuteChange	Numeric	Sets the minutes to be added (positive value) or subtracted (negative value).

Remarks

EasyLanguage does not see time and date as unique data formats but sees them and treats them as normal numeric values.

Example

Assigns to Value1 the calculated time 15 minutes from market close and then plots a PaintBar on every bar after that time:

```
Value1 = CalcTime(1600,-15);
if Time >= Value1 then
 PlotPB(High,Low, "CalcTime");
```

See Also

CalcDate

CCI (Function)

Disclaimer

The CCI function, (Commodity Channel Index) returns the Commodity Channel Index.

Syntax

```
CCI(Length)
```

Returns (Double)

A numeric value for the current bar.

Parameters

Name	Type	Description
Length	Numeric	Sets the number of trailing bars for the CCI to analyze at a time.

Remarks

The CCI value usually does not fall outside the -300 to 300 range and is normally in the -100 to 100 range.

The value for the Length input parameter should always be a whole number greater than 0.

The CCI is calculated by determining the difference between the average price of a commodity and the average of the average prices over some number of bars.

This difference is then compared to the average difference over the same time period, to factor in the commodity's volatility. The result is then multiplied by a constant that is designed to adjust the CCI so that it fits into a normalized range of about +/-100.

Traditionally there are two basic methods of interpreting the CCI, looking for divergences, or treating it as an overbought/oversold oscillator.

Example

Assigns to Value1 and plots a 20 bar CCI value for each bar, then plots Value1:

```
Value1 = CCI(20);
Plot1(Value1, "CCI");
```

Reference

Lamber, Donald R. "Commodity Channel Index" Commodities Magazine. October 1980. Pgs. 40-41.

ChaikinOsc (Function)

Disclaimer

The `ChaikinOsc` series function calculates an oscillator between the fast and slow exponential moving averages of the Accumulated Distribution `(AccumDist)`.

Syntax

```
ChaikinOsc(AnyVol, ShortLength, LongLength)
```

Returns (Double)

A numeric value for the current bar.

Parameters

Name	Type	Description
`AnyVol`	Numeric	Sets the volume (Volume or Ticks) for one of the symbols in the chart.
`ShortLength`	Numeric	Sets the number of bars for the short/fast exponential average.
`LongLength`	Numeric	Sets the number of bars for the long/slow exponential average.

Remarks

The `AnyVol` input parameter should be set to `Volume` when referencing daily data and to `Ticks` when referencing tick/minute data.

The value for the `ShortLength` and `LongLength` input parameters should always be a whole number greater than 0.

Example

Assigns a `ChaikinOsc` value based on the daily Volume to `Value1`, with a short length of 3 and a long length of 10; then plots `Value1`.

```
Value1 = ChaikinOsc (Volume, 3,10);
Plot1(Value1, "ChaikinOsc");
```

Reference

Colby, Robert W. and Thomas A. Meyers. *The Encyclopedia of Technical Market Indicators*. Dow Jones - Irwin. Homewood, IL. 1988.

CloseD (Series Function)

Disclaimer

The `CloseD` function allows you to reference the daily Close of previous days in an intraday chart (minute or tick-based) or a daily chart.

`CloseD` is a function in a family of functions that allows historical daily, weekly, monthly, and yearly references in intraday charts.

Syntax

```
CloseD(PeriodsAgo)
```

Returns (Double)

A numeric value for the current bar. If the `PeriodsAgo` parameter is out of range (> 50), or there is not enough data, the function will return –1.

Parameters

Name	Type	Description
`PeriodsAgo`	Numeric	The number of days/periods back to reference a previous day's closing price. (50 days back maximum) (0 = Today's current Close)

Remarks

You must have enough intraday data in the chart in order to look back and reference any previous close. For example, if you want to look back at the close of 25 days ago on a 5-minute chart, you must have 25 days of 5-minute bars in the chart.

The value for the `PeriodsAgo` input parameter should always be a whole number greater than or equal to 0, but less than 51. Setting `PeriodsAgo` to 0 equals today's current Close.

Example

Assigns the `Close` of the previous day on an intraday chart to `Value1`, then plots `Value1`:

```
Value1 = CloseD(1);
Plot1(Value1, "PrevClose");
```

See Also

HighD, LowD, OpenD, CloseW, CloseM, and CloseY.

CloseM (Series Function)

Disclaimer

The CloseM function allows you to reference the monthly Close of previous months in an intraday chart (minute or tick-based), or a daily, weekly, or monthly chart

CloseM is a function in a family of functions that allows historical daily, weekly, monthly, and yearly references in intraday charts.

Syntax

```
CloseM(PeriodsAgo)
```

Returns (Double)

A numeric value for the current bar. If the PeriodsAgo parameter is out of range (> 50) or there is not enough data, the function will return –1.

Parameters

Name	Type	Description
PeriodsAgo	Numeric	The number of days/periods back to reference a previous day's closing price. (50 months back maximum) (0 = This month's current Close)

Remarks

With intraday charts, you must have enough intraday data in the chart in order to look back and reference any previous close. For example, if you want to look back at the close of 3 months ago on a 5-minute chart, you must have 3 months of 5-minute bars in the chart.

The value for the PeriodsAgo input parameter should always be a whole number greater than or equal to 0, but less than 51. Setting PeriodsAgo to 0 equals today's current Close.

Example

Assigns the Close of the previous month on an intraday chart to Value1, then plots Value1:

```
Value1 = CloseM(1);
Plot1(Value1, "PrevMClose");
```

See also

HighM, LowM, OpenM, CloseD, CloseW, and CloseY.

CloseW (Series Function)

Disclaimer

The CloseW function allows you to reference the weekly Close of previous weeks in an intraday chart (minute or tick-based), or a daily or weekly chart.

CloseW is a function in a family of functions that allows historical daily, weekly, monthly, and yearly references in intraday charts.

Syntax

```
CloseW(PeriodsAgo)
```

Returns (Double)

A numeric value for the current bar. If the PeriodsAgo parameter is out of range (> 50) or there is not enough data, the function will return –1.

Parameters

Name	Type	Description
PeriodsAgo	Numeric	The number of days/periods back to reference a previous day's closing price. (50 weeks back maximum) (0 = This week's current Close)

Remarks

With intraday charts, you must have enough intraday data in the chart in order to look back and reference any previous close. For example, if you want to look back at the close of 3 weeks ago on a 5-minute chart, you must have 3 weeks of 5-minute bars in the chart.

The value for the PeriodsAgo input parameter should always be a whole number greater than or equal to 0, but less than 51. Setting PeriodsAgo to 0 equals today's current Close.

Example

Assigns the Close of the previous week on an intraday chart to Value1, then plots Value1:

```
Value1 = CloseW(1);
Plot1(Value1, "PrevWClose");
```

See also

HighW LowW OpenW CloseD CloseM and CloseY.

CloseY (Series Function)

Disclaimer

The CloseY function allows you to reference the yearly Close of previous years in an intraday chart (minute or tick-based), or a daily, weekly, or monthly chart.

CloseY is a function in a family of functions that allows historical daily, weekly, monthly, and yearly references in intraday charts.

Syntax

```
CloseY(PeriodsAgo)
```

Returns (Double)

A numeric value for the current bar. If the PeriodsAgo parameter is out of range (> 50), or there is not enough data, the function will return –1.

Parameters

Name	Type	Description
PeriodsAgo	Numeric	The number of years/periods back to reference a previous year's closing price. (50 years back maximum) (0 = This year's current Close)

Remarks

With intraday charts, you must have enough intraday data in the chart in order to look back and reference any previous close. For example, if you want to look back at the close of 1 year ago on a 5-minute chart, you must have 1 year of 5-minute bars in the chart.

The value for the PeriodsAgo input parameter should always be a whole number greater than or equal to 0, but less than 51. Setting PeriodsAgo to 0 equals today's current Close.

Example

Assigns the Close of the previous year on an intraday chart to Value1, then plots Value1:

```
Value1 = CloseY(1);
Plot1(Value1, "PrevYClose");
```

See also

HighY, LowY, OpenY, CloseD, CloseW, and CloseM.

CoefficientR (Function)

Disclaimer

The CoefficientR function calculates the Pearson product moment correlation coefficient, *R*.

The Pearson product coefficient, *R*, is a dimensionless index that ranges from -1.0 to 1.0 inclusive and reflects the extent of a linear relationship between Price and a dependent value for a specified period.

Pearson's product moment correlation coefficient, usually denoted by *R*, is one example of a correlation coefficient. It is a measure of the linear association between two variables that have been measured on interval or ratio scales.

A correlation coefficient is a number between -1 and 1, which measures the degree to which two variables are linearly related. If there is a perfect linear relationship with positive slope between the two variables, we have a correlation coefficient of 1; if there is a positive correlation, whenever one variable has a high (low) value, the other does also. If there is a perfect linear relationship with negative slope between the two variables, we have a correlation coefficient of -1; if there is a negative correlation, whenever one variable has a high (low) value, the other has a low (high) value. A correlation coefficient of 0 means that there is no linear relationship between the variables.

Syntax

```
CoefficientR(Indep,Dep,Length)
```

Returns (Double)

A numeric value between 1 and –1, for the current bar. If no value is calculated, the function returns (-2).

Parameters

Name	Type	Description
Indep	Numeric	Specifies which independent bar value (price, function, or formula) to be considered when calculating coefficient R.
Dep	Numeric	Specifies which dependent bar value (price, function, or formula) to be considered when calculating coefficient R.
Length	Numeric	Set the number of bars to consider in the calculation of R.

Remarks

Under certain circumstances the function may not calculate a value for CoefficientR for the current bar. When this happens the function returns –2. You may want to check for this value and not plot for the current bar. Also, if you are looking for negative correlation you may want to add range checking. For example, if *R* < -.75 AND *R* > -1.01 then …

The value for the Length input parameter should always be a whole number greater than 0.

Example

Assigns the CoefficientR of the Close to BarNumber over 20 bars to Value1, then plots a ShowMe on the High when Value1 is greater than .75:

```
Value1 = CoefficientR(BarNumber, Close, 20);
if Value1 > .75 then
Plot1(High, "CoefR");
```

CoefficientRArray (Function)

Disclaimer

The CoefficientRArray function calculates the Pearson product moment correlation coefficient, *R* for two named arrays.

Syntax

```
CoefficientRArray(IndDepArray,DepArray,Size)
```

Returns (Double)

A numeric value between 1 and –1, for two named arrays, for the current bar. If no value is calculated for CoefficientRArray the function returns -2.

Parameters

Name	Type	Description
IndDepArray	Numeric Array	Specifies the name of a numeric array of the independent values.
DepArray	Numeric Array	Specifies the name of a numeric array of the dependent values.
Size	Numeric	Sets the number of data elements (size) in the arrays to be evaluated.

Remarks

The Pearson product coefficient, *R*, is a dimensionless index that ranges from -1.0 to 1.0 inclusive and reflects the extent of a linear relationship between Price and a dependent value for some number of values.

Pearson's product moment correlation coefficient, usually denoted by *R*, is one example of a correlation coefficient. It is a measure of the linear association between two variables that have been measured on interval or ratio scales.

A correlation coefficient is a number between -1 and 1, which measures the degree to which two variables are linearly related. If there is a perfect linear relationship with positive slope between the two variables, we have a correlation coefficient of 1; if there is a positive correlation, whenever one variable has a high (low) value, the other does also. If there is a perfect linear relationship with negative slope between the two variables, we have a correlation coefficient of -1; if there is a negative correlation, whenever one variable has a high (low) value, the other has a low (high) value. A correlation coefficient of 0 means that there is no linear relationship between the variables.

Under certain circumstances the function may not calculate a value for CoefficientRArray for the current bar. When this happens the function returns –2. You may want to check for this value and not plot for the current bar. Also, if you are looking for negative correlation you may want to add range checking. For example if *R* < -.75 AND *R* > -1.01 then …

The value for the Size input parameter should be a whole number greater than 0. Also, Size cannot be larger than the number of elements in the smallest array.

All array-based function calculations begin with array element 1.

Example

Assigns to Value1 the CoefficientRArray of one 10 element array to another 10 element array, then plots a ShowMe on the High when Value1 is greater than .75:

```
Array: myIndDepArray [10](0), myDepArray[10](0);
{… (assign values to both arrays) }
Value1 = CoefficientRArray (myIndDepArray, myDepArray, 10);
if Value1 > .75 then
Plot1(High, "CoefAR");
```

CoefficientREasy (Function)

Disclaimer

The CoefficientREasy function calculates the Pearson product moment correlation coefficient, *R* using a single dependent variable and length.

The Pearson product coefficient, *R*, is a dimensionless index that ranges from -1.0 to 1.0 inclusive and reflects the extent of a linear relationship between Price and a dependent value for a specified period.

Pearson's product moment correlation coefficient, usually denoted by *R*, is one example of a correlation coefficient. It is a measure of the linear association between two variables that have been measured on interval or ratio scales.

A correlation coefficient is a number between -1 and 1, which measures the degree to which two variables are linearly related. If there is a perfect linear relationship with positive slope between the two variables, we have a correlation coefficient of 1; if there is a positive correlation, whenever one variable has a high (low) value, the other does also. If there is a perfect linear relationship with negative slope between the two variables, we have a correlation coefficient of -1; if there is a negative correlation, whenever one variable has a high (low) value, the other has a low (high) value. A correlation coefficient of 0 means that there is no linear relationship between the variables.

Syntax

```
CoefficientREasy(Dep,Length)
```

Returns (Double)

A numeric value between 1 and –1, for the current bar. If no value is calculated the function returns (-2).

Parameters

Name	Type	Description
Dep	Numeric	Specifies which dependent bar value (price, function, or formula) to be considered when calculating coefficient R.
Length	Numeric	Sets the number of trailing bars to consider in the calculation of R.

Remarks

Under certain circumstances the function may not calculate a value for CoefficientREasy for the current bar. When this happens the function returns –2. You may want to check for this value and not plot for the current bar. Also, if you are looking for negative correlation you may want to add range checking. For example, if *R* < -.75 AND *R* > -1.01 then …

The value for the Length input parameter should always be a whole number greater than 0.

Example

```
Value1 = CoefficientREasy(Close,20);
If Value1>0.75 then
Plot1(High,"CoefRE");
```

Combination (Function)

Disclaimer

The Combination function calculates the number of unique combination groups for a given set of numbers.

Consider the following set of numbers: (1,2,3,4). There are 6 combinations of unique 2 number groups in the 4 number set; (1,2), (1,3), (1,4), (2,3), (2,4), and (3,4). So, Combination(4, 2) = 6.

Syntax

```
Combination(Num,NumChosen)
```

Returns (Integer)

A positive numeric value containing the total number of combination groups for a set of numbers for the current bar.

Parameters

Name	Type	Description
Num	Numeric	The total number of items in the set to consider.
NumChosen	Numeric	The number of unique items in each group.

Remarks

The value for Num and NumChosen input parameters should be a positive whole number greater than 0.

Example

Assigns the number of unique 3 number groups in a set of 10 numbers to Value1:

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

CopyColumn (Function)

Disclaimer

The CopyColumn function copies a set of data elements in an array from one column to another over a range of rows.

Syntax

```
CopyColumn(PriceArray, ColIndx2, ColIndx3, StartRow, Size1)
```

Returns (Boolean)

Copies the specified data elements in an array from one column to another. The function itself returns `True` when the copy is completed.

Parameters

Name	Type	Description
`MyArray`	Numeric Array	Specifies the name of a numeric two-dimensional array containing the values to be copied.
`ColIndx2`	Numeric	Sets the column to copy to.
`ColIndx3`	Numeric	Sets the column to copy from.
`StartRow`	Numeric	Sets the row that contains the first element to be copied
`Size1`	Numeric	Sets the row that contains the last element to be copied

Remarks

The function copies data elements from one column to another (2nd dimension index position) over a range of 1st dimension indexes (rows).

The value for the `Size` input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The `CopyColumn` function only works with two-dimensional arrays.

Example

Copies the data elements from column 2 to column 1 in the 4th through 6th rows of `MyArray` and assigns `Condition1` to True when completed.

```
Array: myArray[6,2](0);
{… (add EL statements here to assign values to the array) }
Condition1 = CopyColumn(myArray, 2, 1, 4, 6);
```

Correlation (Function)

Disclaimer

The `Correlation` function calculates the correlation coefficient between an independent and dependent data series.

A correlation coefficient is a number between -1 and 1, which measures the degree to which two variables are linearly related. If there is a perfect linear relationship with positive slope between the two variables, we have a correlation coefficient of 1; if there is a positive correlation, whenever one variable has a high (low) value, the other does also. If there is a perfect linear relationship with negative slope between the two variables, we have a correlation coefficient of -1; if there is a negative correlation, whenever one variable has a high (low) value, the other has a low (high) value. A correlation coefficient of 0 means that there is no linear relationship between the variables.

Syntax

```
Correlation(Indep,Dep,Length)
```

Returns (Double)

A numeric value between 1 and –1 for the current bar.

Parameters

Name	Type	Description
`Indep`	Numeric	Specifies which independent bar value (price, function, or formula) to be considered when calculating the correlation.
`Dep`	Numeric	Specifies which dependent bar value (price, function, or formula) to be considered when calculating the correlation.
`Length`	Numeric	Set the number of bars to consider in the correlation coefficient.

Remarks

The input parameters `Indep` and `Dep` can be a bar value such as `Close`, `High`, `Low`, `Open`, or `Volume`. They can also be any mathematical calculation such as `High` + `Low`) / 2, or a numeric function such as `RSI`, `Stochastic`, or `ADX.`

The value for the `Length` input parameter should always be a whole number greater than 0.

Example

Assigns the `Correlation` of 2 symbols over 20 bars to `Value1`; the `Close` of data1 to the `Close` of data2, then plots a ShowMe on the `High` when `Value1` is greater than .75:

```
Value1 = Correlation (Close of Data1, Close of Data2, 20);
if Value1 > .75 then
Plot1(High, "Corr");
```

CorrelationArray (Function)

Disclaimer

The CorrelationArray function calculates the correlation coefficient for two named arrays.

Syntax

```
CorrelationArray(IndDepArray,DepArray,Size)
```

Returns (Double)

A numeric value between 1 and –1 for the current bar.

Parameters

Name	Type	Description
IndDepArray	Numeric Array	Specifies the name of a numeric array of the independent values.
DepArray	Numeric Array	Specifies the name of a numeric array of the dependent values.
Size	Numeric	Sets the number of data elements (size) in the arrays to be evaluated.

Remarks

A correlation coefficient is a number between -1 and 1, which measures the degree to which two variables are linearly related. If there is a perfect linear relationship with positive slope between the two variables, we have a correlation coefficient of 1; if there is a positive correlation, whenever one variable has a high (low) value, the other does also. If there is a perfect linear relationship with negative slope between the two variables, we have a correlation coefficient of -1; if there is a negative correlation, whenever one variable has a high (low) value, the other has a low (high) value. A correlation coefficient of 0 means that there is no linear relationship between the variables.

The value for the Size input parameter should be a whole number greater than 0. Also, Size cannot be larger than the number of elements in the smallest array.

All array-based function calculations begin with array element 1.

Example

Assigns to Value1 the CorrelationArray value of one 10 element array to another 10 element array, then plots a ShowMe on the High when Value1 is greater than .75:

```
Array: myIndDepArray [10](0), myDepArray [10](0);
{… (assign values to both arrays) }
Value1 = CorrelationArray (myIndDepArray, myDepArray, 10);
if Value1 > .75 then
Plot1(High, "CorrA");
```

CountIF (Function)

Disclaimer

The CountIF function counts the number of true custom test condition occurrences over some number of bars.

The CountIF function can be used for data mining and testing historical statistical market action. For example, you can ask questions such as, "How many days over the last month have been up days with falling volume, and on the following day how many days were up or down?"

Syntax

```
CountIF(Test,Length)
```

Returns (Integer)

The number of true test condition occurrences.

Parameters

Name	Type	Description
Test	TrueFalse	Specifies a bar or indicator condition (True/False expression) that must be TRUE.
Length	Numeric	Sets the number of bars to consider for testing the condition.

Remarks

The input parameters Test can be any bar or multi-data Condition such as Close > Open, High[1] < High[2], Close of Data1 - Close of data2 > 0, or Volume >= 100000. They can also be any mathematical calculation such as High + Low) / 2 > Average(TrueRange,10), or a numeric indicator function such as RSI(C,14) < 30, or ADX(C,14) > 25. It can also be another True/False function.

True/False conditions must contain one of the following operators: **=**, **>=**, **<=**, **>**, **<**, **<>**, **crosses above**, or **crosses below**. You can also test multiple conditions in one expression using: **OR**, and **AND**. Use parentheses (**()**)to group conditions logically.

The value for the Length input parameter should always be a positive whole number greater than 0.

Examples

Assigns the number of bars where the close is greater than open for the last 12 bars to Value1, then plots Value1:

```
Value1 = CountIF(Close > Open, 12);
Plot1(Value1, "C > O");
```

Assigns to Value1 the number of bars where the close is greater than the close of one bar ago, and the volume is greater than the volume of one bar ago, for the last 24 bars, then plots Value1:

```
Value1 = CountIF(Close > Close[1] AND Volume > Volume[1], 24);
Plot1(Value1, "C+V+");
```

Covar (Function)

Disclaimer

The Covar function calculates the covariance of two data series.

The covariance of two data sets measures their tendency to vary together. Where the variance is the average of the squared deviation of a data set from its mean, the covariance is the average of the products of the deviations of the data set values from their means.

Syntax

```
Covar(Indep,Dep,Length)
```

Returns (Double)

A numeric value for the current bar.

Parameters

Name	Type	Description
Indep	Numeric	Specifies which independent bar value (price, function, or formula) to be considered when calculating the covariance.
Dep	Numeric	Specifies which dependent bar value (price, function, or formula) to be considered when calculating the covariance.
Length	Numeric	Sets the number of trailing bars to consider in the covariance.

Remarks

The input parameters Indep and Dep can be a bar value such as Close, High, Low, Open, or Volume. They can also be any mathematical calculation such as High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

The value for the Length input parameter should always be a whole number greater than 0.

Example

Assigns to Value1 the covariance of 2 symbols; the Close of data1 to the Close of data 2 over the last 20 bars, then plots Value1.

```
Value1 = Covar(Close of Data1, Close of Data2, 20);
Plot1(Value1, "Covar");
```

CovarArray (Function)

Disclaimer

The CovarArray function calculates the covariance of two named Arrays.

Syntax

```
CovarArray(IndepArray,DepArray,Size)
```

Returns (Double)

The covariance of two user declared arrays.

Parameters

Name	Type	Description
IndDepArray	Numeric Array	Specifies the name of a numeric array of the independent values.
DepArray	Numeric Array	Specifies the name of a numeric array of the dependent values.
Size	Numeric	Sets the number of data elements (size) in the arrays to be evaluated.

Remarks

The covariance of two data sets measures their tendency to vary together. Where the variance is the average of the squared deviation of a data set from its mean, the covariance is the average of the products of the deviations of the data set values from their means.

The value for the Size input parameter should be a whole number greater than 0. Also, Size cannot be larger than the number of elements in the smallest array.

All array-based function calculations begin with array element 1.

Example

Assigns to Value1 the CovarArray value of one 10 element array to another 10 element array.

```
Array: myIndDepArray [10](0), myDepArray [10](0);
{… (assign values to both arrays) }
Value1 = CovarArray (myIndDepArray, myDepArray, 10);
```

CovarEasy (Function)

Disclaimer

The CovarEasy function calculates the covariance of a single dependent variable and length.

The covariance measures their tendency to vary together. Where the variance is the average of the squared deviation of a data set from its mean, the covariance is the average of the products of the deviations of the data set values from their means.

Syntax

```
CovarEasy(Dep,Length)
```

Returns (Double)

A numeric value for the current bar.

Parameters

Name	Type	Description
Dep	Numeric	Sets the dependent value used to calculate the covariance.
Length	Numeric	Sets the number of bars to consider in the covariance.

Remarks

The input parameter Dep can be a bar or multi-data value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as (High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

The value for the Length input parameter should always be a whole number greater than 0.

Example

Assigns to Value1 the covariance the Close over the last 20 bars, then plots Value1:

```
Value1 = CovarEasy(Close, 20);
Plot1(Value1, "CovarEasy");
```

CSI (Function)

Disclaimer

The CSI series function calculates the Commodity Selection Index (CSI) developed by Wells Wilder.

Syntax

```
CSI(MyMargin,MyCommission,Length)
```

Returns (Double)

A numeric value for the current bar.

Parameters

Name	Type	Description
MyMargin	Numeric	Sets the initial margin required to open a position.
MyCommission	Numeric	Sets the round-turn commission expected to be paid.
Length	Numeric	Sets the number of bars to consider in the ADXR calculation.

Remarks

The value for MyMargin, MyCommission, and Length input parameters should be a number greater than 0.

Wilder's approach was to trade commodities with high CSI values relative to other commodities. Because these commodities tend to be highly volatile, they have the potential to make the most money in the shortest period of time. The CSI is calculated using the ADXR component of the DirMovement (Directional Movement) function.

Example

Assigns to Value1 the Commodity Selection Index (CSI) value using a 20 bar directional movement length, for an electronic future, then plots Value1.

```
Value1 = CSI (5550, 8.00, 20);
Plot1(Value1, "CSI");
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

Cum (Function)

Disclaimer

The Cum series function accumulates the values in a data series beginning with the first bar in the chart (BarNumber = 1), up through the current bar.

Syntax

```
Cum(Price)
```

Returns (Double)

A numeric value containing the cumulative total of Price including the current bar.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to accumulate.

Remarks

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

Examples

Assigns to Value1 the accumulated total Volume for each bar on the chart starting from the first bar on the chart, then plots Value1:

```
Value1 = Cum(Volume);
Plot1(Value1, "AccVol");
```

Assigns to Value1 the average accumulated total Volume for each bar on the chart starting from the first bar on the chart, then plots Value1:

```
Value1 = Cum(Volume) / Barnumber;
Plot1(Value1, "AvgAccVol");
```

CurrentAsk (Function)

Disclaimer

The CurrentAsk function returns the real-time inside Ask for the current bar.

Syntax

```
CurrentAsk
```

Returns (Double)

A numeric real-time Ask value for the current bar. The function returns the Close of the bar for all historical bars.

Parameters

None

Remarks

CurrentAsk cannot reference historical Ask data, and returns the Close of the bar for all historical bars in a chart. Keep in mind that symbols that do not trade directly will report 0; (e.g. Market Index Symbols).

If you historically back-test a Strategy where CurrentAsk is used, the results are not based on Ask data for each bar, but on the Close of each bar.

Examples

Assigns the real-time inside CurrentAsk to Value1, and then places a Limit order at that price.

```
Value1 = CurrentAsk;
Buy next bar at Value1 - .02 Limit;
```

Assigns the real-time inside CurrentAsk to Value1, and then compares that to the last trade price.

```
Value1 = CurrentAsk;
If Value1 = Close then
Sell next bar at Value1 Stop;
```

CurrentBid (Function)

Disclaimer

The CurrentBid function returns the real-time inside Bid for the current bar.

Syntax

```
CurrentBid
```

Returns (Double)

A numeric real-time Bid value for the current bar. The function returns the Close of the bar for all historical bars.

Parameters

None

Remarks

CurrentBid cannot reference historical Bid data, and returns the Close of the bar for all historical bars. Keep in mind that symbols that do not trade directly will report 0; (e.g. Market Index Symbols).

If you historically back-test a Strategy where CurrentBid is used, the results are not based on Bid data for each bar, but on the Close of each bar.

Examples

Assigns the real-time inside CurrentBid to Value1, and then places a Limit order at that price.

```
Value1 = CurrentBid;
Buy next bar at Value1 + .01 Limit;
```

Assigns the real-time inside CurrentBid to Value1, and then compares that to the last trade price.

```
Value1 = CurrentBid;
If Value1 = Close then
Sell next bar at Value1 Stop;
```

CurrentSession (Function)

Disclaimer

The CurrentSession function returns the session number of the current bar.

Syntax

```
CurrentSession(SessionType);
```

Returns (Integer)

A numeric value for the current bar.

Parameters

Name	Type	Description
SessionType	Numeric	Sets the type of session to reference; 0 = AutoDetect, 1 = Regular.

Remarks

The input parameter SessionType specifies the type of session information that should be returned. The type parameter may be specified as follows: Auto-Detect Session – [0, AutoSession] - uses whatever session is specified for the data series;Regular or Custom, or Regular Session – [1, RegularSession] - regular session information should always be returned

Examples

Assigns to Value1 the session number of the current bar in the chart for the regular session:

```
Value1 = CurrentSession(1);
```

See Also

CurrentSessionMS

CurrentSessionMS (Function)

Disclaimer

The `CurrentSessionMS` function returns the session number of the current bar for merged session on multi-data charts.

MS - Merged sessions are from the earliest start time of all symbols to the latest end time for all sessions each trading day.

Syntax

```
CurrentSessionMS;
```

Returns (Integer)

A numeric value for the current bar.

Parameters

None

Remarks

There are generally 5 merged sessions per week for most symbols, so 1 is generally Monday, 2 is Tuesday, etc.

Examples

Assigns to `Value1` the session number for the current bar, for all merged session in a multi-data chart.

```
Value1 = CurrentSessionMS;
```

See Also

CurrentSession

.

DailyLosers (Function)

Disclaimer

The DailyLosers function returns the number of losing strategy positions that were taken throughout the date specified by the input TrgtDate.

Syntax

```
DailyLosers(TargetDate_YYMMDD)
```

Returns (Integer)

The number of losing strategy position entries on a date.

Parameters

Name	Type	Description
TargetDate_YYMMDD	Numeric	Sets the target date, entered in the format "YYYMMDD" or a date related function or calculation.

Remarks

The DailyLosers function only works in a Strategy or a called Function within a Strategy, and will not work in any other indicator or analysis technique study type. The function is limited to looking back 10 positions, so the maximum number of DailyLosers that could be reported is 10, if the last ten trades of the target date were losers.

The input parameter TrgtDate can be a constant value such as 1020601 (6-1-2002), where 102 = 2002, 103 = 2003, and so on. It can also reference date related keywords like Date, (which returns the current bar date), and CurrentDate, (which returns the data feed date.)

Examples

Assigns to Value1 the number of losing trades throughout each day in the chart:

```
Value1 = DailyLosers (Date);
```

Assigns to Value1 the number of losing trades throughout the current day only and checks for three losing trades:

```
Value1 = DailyLosers(CurrentDate);
if Value1 = 3 then …
```

See Also

DailyLosers

DailyWinners (Function)

Disclaimer

The DailyWinners function returns the number of winning strategy positions that were taken throughout the specified date.

Syntax

```
DailyWinners(TargetDate_YYYMMDD)
```

Returns (Integer)

The number of winning strategy position entries on a date.

Parameters

Name	Type	Description
TargetDate_YYYMMDD	Numeric	Sets the target date, entered in the format "YYYMMDD" or a date related function or calculation.

Remarks

The DailyWinners function only works in a Strategy or a Function called from within a Strategy and it will not work in any other indicator or analysis technique study type. The function is limited to looking back 10 positions, so the maximum number of DailyWinners that could be reported is 10, if the last ten trades of the target date were winners.

The input parameter TrgtDate can be a constant value like; 1020601 (6-1-2002), where 102 = 2002, 103 = 2003, 99 = 1999, and so on. It can also reference date related keywords like Date (which returns the current bar date), and CurrentDate, (which returns the datafeed date.)

Examples

Assigns to Value1 the number of winning trades throughout each day in the chart:

```
Value1 = DailyWinners (Date);
```

Assigns to Value1 the number of winning trades throughout the current day only and checks for three winning trades:

```
Value1 = DailyWinners (CurrentDate);
if Value1 = 3 then …
```

See Also

DailyWinners

DaysToExpiration (Function)

Disclaimer

The `DaysToExpiration` function returns the number of days left until a specified option expiration date. The calculation is based on the third Friday of every month, which means it's not designed for most commodities, and commodities options.

Syntax

```
DaysToExpiration(ExpMonth,ExpYear)
```

Returns (Integer)

A numeric value for the current bar, containing the number of days until the stock option expires from the current date. This function will return a negative number from the expiration date if the expiration date has already occurred.

Parameters

Name	Type	Description
`ExpMonth`	Numeric	Sets the target month, entered as a number; 1 = January, 2 = February, and so on.
`ExpYear`	Numeric	Sets the target year, entered in the format YYY; 99 = 1999, 100 = 2000, 101 = 2001, 102 = 2002, and so on.

Examples

Assigns to `Value1` the number of days until the stock option expires in December from today:

```
Value1 = DaysToExpiration (12,102);
```

Detrend (Function)

Disclaimer

The Detrend function calculates the detrended value of a price from an offset average of the price.

Syntax

```
Detrend(Price,Length)
```

Returns (Double)

The numeric detrend value for the current bar.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to consider for the detrended average.
Length	Numeric	Sets the number of bars to consider for the detrended average calculation.

Remarks

The detrend average offset is half of the average length plus one.

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

The value for the Length input parameter should always be a whole number greater than 0.

Example

Assigns to Value1 the detrended price value for the Close from the 14 bar average of the Close, then plots Value1:

```
Value1 = Detrend(Close, 14);
Plot1(Value1, "Detrend");
```

DevSqrd (Function)

Disclaimer

The DevSqrd function calculates the sum of squares of deviations of a price from the average.

The calculation of this formula is similar to the classic statistical volatility (Statistical Standard Deviation), the difference being that peaks (higher values) are accentuated because the values are squared.

Syntax

```
DevSqrd(Price,Length)
```

Returns (Double)

A numeric value for the current bar.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Length	Numeric	Sets the number of trailing bars to consider for the calculation.

Remarks

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

The value for the Length input parameter should always be a whole number greater than 0.

Example

Assigns to Value1 the DevSqrd value for the Close looking at the 14 bar average of the Close, then plots Value1:

```
Value1 = DevSqrd(Close, 14);
Plot1(Value1, "DevSqrd");
```

DevSqrdArray (Function)

Disclaimer

The DevSqrdArray function calculates the sum of squares of deviations of values in an array from the average.

Syntax

```
DevSqrdArray(PriceArray,Size)
```

Returns (Double)

A numeric value containing the sum of squares of deviations from an average for an array of values.

Parameters

Name	Type	Description
PriceArray	Numeric	Specifies the name of a numeric array containing values upon which the sum of squares of deviations is calculated.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.

Remarks

The DevSqrdArray calculation is similar to the classic statistical volatility (Statistical Standard Deviation), the difference being that peaks (higher values) are accentuated because the values are squared.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The DevSqrdArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Assigns to Value1 the Skew factor of the specified named array MyArray with 40 elements.

```
Array: MyArray[40](0);
{… add EL statements here to assign price values to array elements... }
Value1 = DevSqrdArray(MyArray,40);
```

DirMovement (Function)

Disclaimer

The DirMovement series function calculates the Directional Movement values described by Welles Wilder.

The DirMovement function is a multiple-output function that consolidates the calculations of several related functions: ADX, ADXCustom, ADXR, ADXRCustom, DMI, DMICustom, DMIMinus, DMIMinusCustom, DMIPlus, and DMIPlusCustom.

Syntax

```
DirMovement(PriceH,PriceL,PriceC,Length,oDMIPlus,oDMIMinus,oDMI,oADX,oADXR,oVolatility)
```

Returns (Integer)

The oDMIPlus, oDMIMinus, oDMI, oADX, oADXR, and oVolty output parameters return the related directional movement values. The DirMovement function itself returns a value of 1.

Parameters

Name	Type	Description
PriceH	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement high price.
PriceL	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement low price.
PriceC	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement close price.
Length	Numeric	Sets the number of bars to consider for the average directional movement index rating calculation.
oDMIPlus	Numeric	Outputs the DMIPlus value.
oDMIMinus	Numeric	Outputs the DMIMinus value.
oDMI	Numeric	Outputs the DMI value.
oADX	Numeric	Outputs the ADX value.
oADXR	Numeric	Outputs the ADXR value.
oVolty	Numeric	Outputs the volatility value.

Remarks

The PriceH, PriceL, and PriceC input parameters will need to reference some High, Low, and Close values based on the symbol(s) in your chart.

The value for the Length input parameter should always be a positive whole number.

See Multiple Output Function for more information on using output parameters to return values.

Example

Assigns to Value2 the calculated 14 bar ADX value passed by reference to the oADX parameter, and assigns to Value3 the calculated 14 bar ADXR value passed by reference to the oADXR. Value1 is assigned the value of 1:

```
Value1 = DirMovement (High, Low, Close, 14,
    oDMIPlus, oDMIMinus, oDMI, oADX, oADXR, oVolatility);
Value2 = oADX;
Value3 = oADXR;
```

Note This example only uses 2 of the 6 output parameters, although the other output values are still valid.

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

Colby, Robert F., Myers, Thomas. A. *The Encyclopedia of Technical Market Indicators*. Dow Jones – Irwin. Homewood, IL. 1988.

Murphy, John J. *The Visual Investor*. New York, NY: John Wiley & Sons, Inc. 1996.

Divergence (Function)

Disclaimer

The Divergence function identifies occurrences of divergences in two Pivot high or low conditions for the BullishDivergence and the BearishDivergence functions.

A BullishDivergence is normally identified when a price (Low) makes a Pivot low that is lower than the previous Pivot low, and an oscillator (Stochastic, RSI, BollingerBands) makes a Pivot low that is higher than the previous Pivot low.

A BearishDivergence is normally identified when a price (High) makes a Pivot high that is higher than the previous Pivot high, and an oscillator (Stochastic, RSI, BollingerBands) makes a Pivot high that is lower than the previous Pivot high.

Syntax

```
Divergence(Price1,Price2,Strength,Length,HiLo)
```

Returns (Integer)

The function returns a positive one (1) if the divergence condition is found true for the current bar and returns a zero (0) if the divergence condition is not found for the current bar.

Parameters

Name	Type	Description
Price1	Numeric	Specifies which bar value (price, function, or formula) to use for the first pivot point.
Price2	Numeric	Specifies which bar value (price, function, or formula) to use for the second pivot point.
Strength	Numeric	Sets the number of bars on either side of the pivot high point.
Length	Numeric	Sets the maximum number of trailing bars between pivot high points.
HiLo	Numeric	Set the type of divergence to calculate. -1 = bullish divergence, 1 = bearish divergence.

Remarks

The input parameters Price1 and Price2 can be a bar value such as Close, High, Low, Open, or Volume. They can also be any mathematical calculation such as High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

Increasing the Strength input parameter will normally decrease the number of True occurrences.

Increasing the Length input parameter will normally increase the chances for True of occurrences.

The value for the Length and Strength input parameters should always be a whole number greater than 0.

The HiLo input parameter is a switch, when HiLo = -1, the bullish divergence is identified, when HiLo = 1, the bearish divergence is identified.

Example

Plots a ShowMe on the High when a bearish divergence occurs between a Pivot High and the Pivot high of the RSI oscillator on any bar, where the Pivot Strength is 2 and the Pivot Length is 20:

```
if Divergence(High, RSI(Close,14), 2, 20, 1) = 1 then
 Plot1(High, "BearDrvg");
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

See Also

BearishDivergence, BullishDivergence

DMI (Function)

Disclaimer

The DMI series function returns the directional movement index (DMI) for a security.

Syntax

```
DMI(Length)
```

Returns (Double)

A positive numeric value for the current bar.

Parameters

Name	Type	Description
Length	Numeric	Sets the number of bars to consider for the directional movement index calculation.

Remarks

DMI attempts to measure the trending quality of a security independent of direction. The greater the DMI value, the stronger a security is trending. DMI does not indicate direction.

The DirMovement function calculates the DMI value.

The value for the Length input parameter should always be a whole number greater than 0.

Example

Assigns to Value1 the DMI value where the DMI length is 14 bars, then plots Value1:

```
Value1 = DMI(14);
Plot1(Value1, "DMI");
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

See Also

DMICustom, DMIMinus, DMIPlus, DirMovement

DMICustom (Function)

Disclaimer

The DMICustom series function returns the directional movement index (DMI) for a security.

Syntax

```
DMICustom(PriceH,PriceL,PriceC,Length)
```

Returns (Double)

A positive numeric value for the current bar.

Parameters

Name	Type	Description
PriceH	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement high price.
PriceL	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement low price.
PriceC	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement close price.
Length	Numeric	Sets the number of bars to consider for the directional movement index plus calculation.

Remarks

DMICustom attempts to measure the trending quality of a security independent of direction. The greater the DMICustom value, the stronger a security is trending. DMICustom does not indicate direction.

DMICustom provides added flexibility over the normal DMI function by allowing you to specify what High, Low, and Close values to use. This is generally used to reference multi-data elements other than Data1.

The DirMovement function calculates the DMICustom value.

The value for the Length input parameter should always be a whole number greater than 0.

Example

Assigns to Value1 the DMICustom value of the second data element in a chart (Data2), High of Data2, Low of Data2, Close of Data2, where the average length is 14 bars, then plots Value1:

```
Value1 = DMICustom(High of Data2, Low of Data2, Close of Data2,14);
Plot1(Value1, "DMICus");
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

See Also

DMI, DirMovement

DMIMinus (Function)

Disclaimer

he DMIMinus series function returns the directional movement index minus value for a security.

Syntax

```
DMIMinus(Length)
```

Returns (Double)

A positive numeric value for the current bar.

Parameters

Name	Type	Description
Length	Numeric	Sets the number of bars to consider for the directional movement index plus calculation.

Remarks

DMIMinus measures downward movement strength.

DMIMinus is the one of the core components to the directional movement calculations. It is derived by comparing the Low of the current bar to the Low of the previous bar. If the Low of the current bar is less than the Low of the previous bar, than DMIMinus value is that difference for the current bar.

The DirMovement function calculates the DMIMinus value.

The value for the Length input parameter should always be a whole number.

Example

Assigns to Value1 the DMIMinus value where the DMI length is 14 bars, then plots Value1:

```
Value1 = DMIMinus(14);
Plot1(Value1, "DMIMinus");
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

See Also

DMIPluseCustom, DMIPlus, DMI, DirMovement

DMIMinusCustom (Function)

Disclaimer

The DMIMinusCustom series function returns the directional movement index minus value for a security.

Syntax

```
DMIMinusCustom(PriceH,PriceL,PriceC,Length)
```

Returns (Double)

A positive numeric value for the current bar.

Parameters

Name	Type	Description
PriceH	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement high price.
PriceL	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement low price.
PriceC	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement close price.
Length	Numeric	Sets the number of bars to consider for the directional movement index plus calculation.

Remarks

DMIMinusCustom measures downward movement strength.

DMIMinusCustom is the one of the core components to the directional movement calculations. It is derived by comparing the Low of the current bar to the Low of the previous bar. If the Low of the current bar is less than the Low of the previous bar, than DMIMinus value is that difference for the current bar.

DMIMinusCustom provides added flexibility over the normal DMIMinus function by allowing you to specify what High, Low, and Close values to use. This is generally used to reference multi-data elements other than Data1.

The DirMovement function calculates the DMIMinusCustom value.

The value for the Length input parameter should always be a whole number greater than 0.

Example

Assigns to Value1 the DMIMinusCustom value of the second data element in a chart (Data2), High of Data2, Low of Data2, Close of Data2, where the DMI length is 14 bars, then plots Value1:

```
Value1 = DMIMinusCustom (High of Data2, Low of Data2, Close of Data2,14);
Plot1(Value1, "DMIMCus");
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

See Also

DMIMinus, DMI, DirMovement, DMIPlus

DMIPlus (Series Function)

Disclaimer

The DMIPlus series function returns the directional movement index plus value for a security.

Syntax

```
DMIPlus(Length)
```

Returns (Double)

A positive numeric value for the current bar.

Parameters

Name	Type	Description
Length	Numeric	Sets the number of bars to consider for the directional movement index plus calculation.

Remarks

The value for the Length input parameter should always be a whole number greater than 0.

DMIPlus measures upward movement strength.

DMIPlus is the one of the core components to the directional movement calculations. It is derived by comparing the High of the current bar to the High of the previous bar. If the High of the current bar is greater than the High of the previous bar, than DMIPlus value is that difference for the current bar.

The DirMovement function calculates the DMIPlus value.

Example

Assigns to Value1 the DMIPlus value where the DMI length is 14 bars, then plots Value1:

```
Value1 = DMIPlus(14);
Plot1(Value1, "DMIPlus");
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

See Also

DMIPluseCustom, DMIMinus, DMI, DirMovement

DMIPlusCustom (Series Function)

Disclaimer

The DMIPlusCustom series function returns the directional movement index plus value for a security.

Syntax

```
DMIPlusCustom(PriceH,PriceL,PriceC,Length)
```

Returns (Double)

A positive numeric value for the current bar.

Parameters

Name	Type	Description
PriceH	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement high price.
PriceL	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement low price.
PriceC	Numeric	Specifies which bar value (price, function, or formula) to use for the directional movement close price.
Length	Numeric	Sets the number of bars to consider for the directional movement index plus calculation.

Remarks

DMIPlusCustom measures upward movement strength.

DMIPlusCustom is the one of the core components to the directional movement calculations. It is derived by comparing the High of the current bar to the High of the previous bar. If the High of the current bar is greater than the High of the previous bar, than DMIPlusCustom value is that difference for the current bar.

DMIPlusCustom provides added flexibility over the normal DMIPlus function by allowing you to specify what High, Low, and Close values to use. This is generally used to reference multi-data elements other than Data1.

The DirMovement function calculates the DMIPlusCustom value.

The PriceH, PriceL, and PriceC input parameters will need to reference some High, Low, and Close values based on the symbol(s) in your chart.

The value for the Length input parameter should always be a positive whole number.

Example

Assigns to Value1 the DMIPlusCustom value of the second data element in a chart (Data2), High of Data2, Low of Data2, Close of Data2, where the DMI length is 14 bars, then plots Value1:

```
Value1 = DMIPlusCustom (High of Data2, Low of Data2, Close of Data2,14);
Plot1(Value1, "DMIPCus");
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC.

See Also

DMIPlus, DMIMinus, DMI, DirMovement

EaseOfMovement (Function)

Disclaimer

The EaseOfMovement function calculates the relationship between volume and price change using the Market Facilitation Index (MFI) function.

Syntax

```
EaseOfMovement
```

Returns (Double)

A positive or negative number for the current bar.

Parameters

None.

Remarks

The EaseOfMovement function tries to gauge how easily prices are moving up or down and is usually smoothed with a moving average.

High positive EaseOfMovement values occur when prices are moving upward on light volume. Low negative EaseOfMovement values occur when prices are moving downward on light volume. If prices are not moving, or if heavy volume is required to move prices, then the indicator will be near zero.

Example

Assigns to Value1 and plots the 10 bar average of the EaseOfMovement value:

```
Value1 = Average(EaseOfMovement, 10);
Plot1(Value1, "EOM");
```

Reference

Richard W. Arms, Jr. developed the Ease of Movement concept.

Colby, Robert W. and Thomas A. Meyers. *The Encyclopedia of Technical Market Indicators*. Dow Jones - Irwin. Homewood, IL. 1988.

ELDate (Function)

Disclaimer

The ELDate function returns a date in EasyLanguage format (YYYMMDD) based on a specified month, date, and year.

Syntax

```
ELDate(MM,DD,YYYY)
```

Returns (Integer)

A numeric value that represents the date in EasyLanguage format for the current bar.

Parameters

Name	Type	Description
MM	Numeric	Sets the two digit month code: 01 = January, 12 = December, and so on.
DD	Numeric	Sets the two digit date code: 01 = 1st of the month, 30 = the 30th of the month, and so on.
YYYY	Numeric	Set the four-digit year code: 1998, 2002, 2006, and so on.

Remarks

The values for MM, DD, or YYYY input parameters should always be whole numbers representing valid calendar dates.

For example, in EasyLanguage format, 1020704 = July 4th, 2002, or 991225 = December 25, 1999, where the year is represented as 103 for 2003, 102 for 2002, 101 for 2001, 100 for 2000, 99 for 1999, and so on.

The reserved words Date and CurrentDate return numeric date values in this same EasyLanguage format (YYYMMDD).

Example

Assigns to Value1 the numeric EasyLanguage date of 1020704, for the input parameters of; 07,04,2002, (July, 4th, 2002):

```
Value1 = ELDate(07, 04, 2002);
```

ELDate_Consol (Function)

Disclaimer

The ELDate_Consol function translates a YYYYMMDD date to EasyLanguage format (YYYMMDD).

Syntax

```
ELDate_Consol(YYYYMMDD)
```

Returns (Integer)

A date in EasyLanguage format.

Parameters

Name	Type	Description
YYYYMMDD	Numeric	Sets a YYYYMMDD date to be translated. Enter 20060714 for July, 14, 2006.

Remarks

The value for YYYYMMDD should always be a whole number representing valid calendar dates containing a 4-digit year, 2-digit month, and 2-digit year.

For example, in EasyLanguage format, 1020704 = July 4th, 2002, or 991225 = December 25, 1999, where the year is represented as 103 for 2003, 102 for 2002, 101 for 2001, 100 for 2000, 99 for 1999, and so on.

The reserved words Date and CurrentDate return numeric date values in this same EasyLanguage format (YYYMMDD).

Example

Assigns to Value1 the numeric EasyLanguage date of 1020704 from the input date 20020704 (July, 4th, 2002).

```
Value1 = ELDate_Consol(20020704);
```

ELDateToString (Function)

Disclaimer

The `ELDateToString` function returns a 10-character date string in MM/DD/YYYY format from the EasyLanguage date specified. Example: 1020704 = "07/04/2002"

Syntax

```
ELDateToString(DateSelect)
```

Returns (String)

A text string containing a 10-character date in MM/DD/YYYY format for the current bar.

Parameter

Name	Type	Description
DateSelect	Numeric	Sets the target date, entered in the EL format "YYYMMDD" or a numeric date related function or calculation.

Remarks

The value for the `DateSelect` input parameter can be any valid date (past or future) in EasyLanguage format.

For example, in EasyLanguage format, 1020704 = July 4th, 2002, or 991225 = December 25, 1999, where the year is represented as 103 for 2003, 102 for 2002, 101 for 2001, 100 for 2000, 99 for 1999, and so on.

The reserved words `Date` and `CurrentDate` return numeric date values in EasyLanguage format (YYYMMDD).

Example

Assigns to `Value1` the text string for the numeric EasyLanguage date of 1020704 (July, 4th, 2002):

```
Value1 = ELDateToString(1020704);
```

Assigns to `Value1` the text string for the current datafeed time:

```
Value1 = ELDateToString(CurrentDate);
```

Assigns to `Value1` the text string for the current bar date:

```
Value1 = ELDateToString(Date);
```

EntriesToday (Function)

Disclaimer

The EntriesToday function returns the number of strategy position entries that were taken throughout the specified date.

Syntax

```
EntriesToday(TargetDate_YYYYMMDD)
```

Returns (Integer)

The number of strategy position entries on a date.

Parameters

TargetDate_YYYMMDD	Sets the target date, entered in the format "YYYMMDD" or a date related function or calculation.

Remarks

The EntriesToday function only works in Strategies, and will not work in any other indicator or analysis technique study type. The function is limited to looking back 10 positions, so the maximum number of EntriesToday that could be reported is 10, if there were 10 entries today.

The input parameter can be a constant value like such as 1020601 (6-1-2002), where 102 = 2002, 103 = 2003, 99 = 1999, and so on. It can also reference date related keywords like Date (which returns the current bar date) and CurrentDate (which returns the datafeed date).

Examples

Assigns to Value1 the number of entries throughout each day on the chart:

```
Value1 = EntriesToday(Date);
```

Assigns to Value1 the number of entries throughout the current day only and then checks for three entries:

```
Value1 = EntriesToday(CurrentDate);
if Value1 = 3 then …
```

See Also

ExitsToday

ExitsToday (Function)

Disclaimer

The ExitsToday function returns the number of strategy position exits that occurred throughout the date specified by the input parameter Date0.

Syntax

```
ExitsToday(TargetDate_YYMMDD)
```

Returns (Integer)

The number of strategy position exits on a date.

Parameters

TargetDate_YYYMMDD	Sets the target date, entered in the format "YYYMMDD" or a date related function or calculation.

Remarks

The ExitsToday function only works in Strategies, and will not work in any other indicator or analysis technique study type. The function is limited to looking back 10 positions, so the maximum number of ExitsToday that could be reported is 10, if there were 10 entries today.

The input parameter can be a constant value like such as 1020601 (6-1-2002), where 102 = 2002, 103 = 2003, 99 = 1999, and so on. It can also reference date related keywords like Date (which returns the current bar date) and CurrentDate (which returns the datafeed date).

Examples

Assigns to Value1 the number of exits throughout each day on the chart:

```
Value1 = ExitsToday (Date);
```

Assigns to Value1 the number of exits throughout the current day only and then checks for three exits:

```
Value1 = ExitsToday (CurrentDate);
if Value1 = 3 then …
```

See Also

EntriesToday

ExtremePriceRatio (Simple Function)

Disclaimer

The ExtremePriceRatio function calculates the ratio of the extreme prices for a specified number of trailing bars. The extreme prices used are the Highest High and Lowest Low over a specified number of bars.

Ratio = x/y, where x is the Highest High and y is the Lowest Low in the range of bars.

Syntax

```
ExtremePriceRatio(Length,UseLog)
```

Returns (Double)

A numeric value for the current bar.

Parameters

Length	A numeric value. The number of trailing bars to consider for the calculation range.
UseLog	True/false expression. TRUE = Use a Log of the ratio calculation, FALSE = Use a simple ratio calculation.

Remarks

If the input parameter Length were set to 1, the ratio of the current bar's High and Low would be returned.

The value for the Length should always be a whole number greater than 0.

The input parameter UseLog is a switch; it allows you to select the type of calculation to be performed. When TRUE a Log of the ratio of extreme prices is calculated. When FALSE a simply ratio of the extreme prices is calculated.

Example

Assigns to Value1 and plots the simple ratio of extreme prices for the trailing 10 bars:

```
Value1 = ExtremePriceRatio(10, FALSE);
Plot1(Value1, "EPR");
```

Assigns to Value1 and plots the logarithmic ratio of extreme prices for the trailing 10 bars:

```
Value1 = ExtremePriceRatio(10, TRUE);
Plot1(Value1, "LogEPR");
```

Extremes (Function)

Disclaimer

The Extremes function returns the extreme highest or lowest value over a range of bars and how many bars ago the extreme value occurred. There may be times when two or more bars have the exact same extreme highest or lowest value; when this happens the function identifies the most recent occurrence.

Syntax

```
Extremes(Price,Length,HiLo,oExtremeVal,oExtremeBar)
```

Returns (Integer)

The oExtremeVal and oExtremeBar output parameters return the extreme value and the number of bars ago it occurred. The Extremes function itself returns a value of 1.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to compare for highest and lowest extremes.
Length	Numeric	Sets the number of bars to consider for extremes.
HiLo	Numeric	Sets whether the function will return the highest or lowest extreme value. 1=Highest, -1=Lowest.
oExtremeVal	Numeric	Outputs the highest or lowest extreme value found for the range of bars based on the HiLo setting.
oExtremeBar	Numeric	Outputs the number of bars ago the extreme value occurred.

Remarks

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as: (High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

See Multiple Output Function for more information on using output parameters to return values.

Example

Assigns to Value2 the highest High of the last 20 bars using the oExtremeVal output parameter, and assigns to Value3 the number of bars ago the highest High occurred using oExtremeBar output parameter. Value1 is assigned a value of 1:

```
vars: oExtremeVal(0), oExtremeBar(0);
Value1 = Extremes(High, 20, 1, oExtremeVal, oExtremeBar);
Value2 = oExtremeVal;
Value3 = oExtremeBar;
```

See Also

ExtremesFC, ExtremesArray, NthExtremes.

ExtremesArray (Function)

Disclaimer

The ExtremesArray function returns the extreme highest or lowest value from an Array of values and how many bars ago the extreme value occurred. There may be times when two or more bars have the exact same extreme highest or lowest value; when this happens the function identifies the most recent occurrence.

Syntax

```
ExtremesArray(PriceArray,Size,HiLo,oExtremeVal,oExtremePosRaw)
```

Returns (Integer)

The oExtremeVal and oExtremeBar output parameters return the extreme value and the number of bars ago it occurred. The ExtremesArray function itself returns a value of 1.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing bar values to compare for highest and lowest extremes.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.
HiLo	Numeric	Sets whether the function will return the highest or lowest extreme value. 1=Highest, -1=Lowest.
oExtremeVal	Numeric	Outputs the highest or lowest extreme value found for the range of bars based on the HiLo setting.
oExtremePosRaw	Numeric	Outputs the number of bars ago the extreme value occurred.

Remarks

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The ExtremesArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

See Multiple Output Function for more information on using output parameters to return values.

Note This array function is similar to Extremes except that you can build an array of values that does not necessarily include contiguous bar values.

Example

Assigns to Value2 the highest High in the specified named array myArray with 20 elements using the oExtremeVal output parameter, and assigns to Value3 the number of bars ago the highest High occurred using oExtremeBar output parameter. Value1 is assigned a value of 1:

```
Array: myArray[20](0);
Vars: oExtremeVal(0), oExtremePosRaw(0);
{… add EL statements here to assign High values to array elements... }
Value1 = ExtremesArray (myArray, 20, 1, oExtremeVal, oExtremePosRaw);
Value2 = oExtremeVal;
Value3 = oExtremePosRaw;
```

See Also

Extremes, NthExtremesArray.

ExtremesFC (Function)

Disclaimer

The ExtremesFC (Fast Calculation) series function returns the extreme highest or lowest value over a range of bars and how many bars ago the extreme value occurred. There may be times when more two or more bars have the exact same extreme highest or lowest value; when this happens the function identifies the most recent occurrence.

Syntax

```
ExtremesFC(Price,Length,HiLo,oExtremeVal,oExtremeBar)
```

Returns (Integer)

The oExtremeVal and oExtremeBar output parameters return the extreme value and the number of bars ago it occurred. The ExtremesFC function itself returns a value of 1.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to compare for highest and lowest extremes.
Length	Numeric	Sets the number of bars to consider for extremes.
HiLo	Numeric	Sets whether the function will return the highest or lowest extreme value. 1=Highest, -1=Lowest.
oExtremeVal	Numeric	Outputs the highest or lowest extreme value found for the range of bars based on the HiLo setting.
oExtremeBar	Numeric	Outputs the number of bars ago the extreme value occurred.

Remarks

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as: (High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

See Multiple Output Function for more information on using output parameters to return values.

Note This series function returns exactly the same values as Extremes except that it uses a fast calculation method that takes slightly more memory than the non-FC version.

Example

Assigns to Value2 the highest High of the last 20 bars using the oExtremeVal output parameter, and assigns to Value3 the number of bars ago the highest High occurred using oExtremeBar output parameter. Value1 is assigned a value of 1:

```
vars: oExtremeVal(0), oExtremeBar(0);
Value1 = ExtremesFC(High, 20, 1, oExtremeVal, oExtremeBar);
Value2 = oExtremeVal;
Value3 = oExtremeBar;
```

See Also

Extremes, ExtremesArray

Factorial (Function)

Disclaimer

Calculates the factorial of a number, Num, by multiplying all positive integers less then or equal to Num.

Syntax

```
Factorial(Num)
```

Parameters

Name	Type	Description
Num	Numeric	The base number from which to calculate the factorial.

Returns (Double)

The Factorial of the specified number. Returns an error value of -1 if Num is a negative number.

Usage

The factorial of a number is equal to 1*2*3*...* number. The number needs to be positive, and if it has a decimal value it will be truncated.

Example

In the following expression, Value1 will be equal to 24 (1*2*3*4=24):

```
Value1 = Factorial(4);
```

FastD (Function)

Disclaimer

The `FastD` series function returns the Fast D value for the Stochastic oscillator.

Syntax

```
FastD(StochLength)
```

Returns (Double)

A numeric value containing `FastD` for the current bar.

Parameters

Name	Type	Description
`StochLength`	Numeric	Sets the number of bars to consider.

Remarks

Please refer to the discussion under the Stochastic function.

Example

Assigns to `Value1` the Stochastic FastD over 14 bars.

```
Value1 = FastD(14);
```

Reference

Takano, Mike. *Stochastic Oscillator, Technical Analysis of Stocks and Commodities*. April 1991.

Stein, John. *The Traders' Guide to Technical Indicators, Futures Magazine*. August 1990.

FastDCustom (Function)

Disclaimer

The `FastDCustom` series function returns the fast D value for the Stochastic oscillator based on user-defined price inputs.

Syntax

```
FastDCustom(PriceH, PriceL, PriceC, StochLength)
```

Returns (Double)

A numeric value containing the `FastDCustom` for the current bar.

Parameters

Name	Type	Description
`PriceH`	Numeric	Specifies which bar value (price, function, or formula) to use for the high in stochastic calculations.
`PriceL`	Numeric	Specifies which bar value (price, function, or formula) to use for the low in stochastic calculations.
`PriceC`	Numeric	Specifies which bar value (price, function, or formula) to use for the close in stochastic calculations.
`StochLength`	Numeric	Sets the number of bars to consider.

Remarks

Please refer to the discussion under the Stochastic function.

Example

Assigns to `Value1` the Stochastic FastD for offset high and low prices over 14 bars.

```
Value1 = FastDCustom(High+1,Low-1,Close,14);
```

Reference

Takano, Mike. *Stochastic Oscillator, Technical Analysis of Stocks and Commodities*. April 1991.

Stein, John. *The Traders' Guide to Technical Indicators, Futures Magazine*. August 1990.

FastDCustomOrig (Function)

Disclaimer

The FastDCustomOrig function returns the fast D value for the Stochastic oscillator based on user-defined price inputs.

Syntax

```
FastDCustomOrig(PriceH, PriceL, PriceC, StochLength, SmoothingLength)
```

Returns (Double)

A numeric value containing the FastDCustomOrig for the current bar.

Parameters

Name	Type	Description
PriceH	Numeric	Specifies which bar value (price, function, or formula) to use for the high in stochastic calculations.
PriceL	Numeric	Specifies which bar value (price, function, or formula) to use for the low in stochastic calculations.
PriceC	Numeric	Specifies which bar value (price, function, or formula) to use for the close in stochastic calculations.
StochLength	Numeric	Sets the number of bars to consider.
SmoothingLength	Numeric	Sets the constant to use for smoothing the K line.

Remarks

This function differs from FastDCustom by using the original smoothing method suggested by George Lane.

Please refer to the discussion under the Stochastic function.

Example

Assigns to Value1 the Stochastic FastK for offset high and low prices over 14 bars.

```
Value1 = FastDCustomOrig(High+1,Low-1,Close,14,3);
```

Reference

Takano, Mike. *Stochastic Oscillator, Technical Analysis of Stocks and Commodities*. April 1991.

Stein, John. *The Traders' Guide to Technical Indicators, Futures Magazine*. August 1990.

FastHighestBar (Function)

Disclaimer

The FastHighestBar function uses a fast calculation method to return the highest value found when applying the input Price over a period of time defined by the input Length.

Function

```
FastHighestBar(Price, Length)
```

Parameters

Price	Specifies which price of the asset of interest is to be used.
Length	The number of trailing bars to consider.

Returns

A numeric value containing the number of bars ago that the highest value found occurred.

Usage

The input Price can be hard coded with a bar attribute such as Close, Open, High, Low, and Volume or a numeric series type input. It can also be replaced with a valid EasyLanguage expression. For example: Close + Open, or Average(RSI(Close,14),14).

The input Length, can be hard coded replaced with a numeric simple type input.

Note This function uses a fast calculation method that uses more memory than the traditional method.

FastK (Function)

Disclaimer

The FastK series function returns the Fast K value for the Stochastic oscillator.

Syntax

```
FastK(StochLength)
```

Returns (Double)

A numeric value containing FastK for the current bar.

Parameters

Name	Type	Description
StochLength	Numeric	Sets the number of bars to consider.

Remarks

Please refer to the discussion under the Stochastic function.

Example

Assigns to Value1 the Stochastic FastK over 14 bars.

```
Value1 = FastK(14);
```

Reference

Takano, Mike. *Stochastic Oscillator, Technical Analysis of Stocks and Commodities*. April 1991.

Stein, John. *The Traders' Guide to Technical Indicators, Futures Magazine*. August 1990.

FastKCustom (Function)

Disclaimer

The `FastKCustom` function returns the Fast K value for the Stochastic oscillator based on user-defined price inputs.

Syntax

```
FastKCustom(PriceH, PriceL, PriceC, StochLength)
```

Returns (Double)

A numeric value containing the `FastKCustom` for the current bar.

Parameters

Name	Type	Description
`PriceH`	Numeric	Specifies which bar value (price, function, or formula) to use for the high in stochastic calculations.
`PriceL`	Numeric	Specifies which bar value (price, function, or formula) to use for the low in stochastic calculations.
`PriceC`	Numeric	Specifies which bar value (price, function, or formula) to use for the close in stochastic calculations.
`StochLength`	Numeric	Sets the number of bars to consider.

Remarks

Please refer to the discussion under the Stochastic function.

Example

Assigns to `Value1` the Stochastic FastK for offset high and low prices over 14 bars.

```
Value1 = FastKCustom(High+1,Low-1,Close,14);
```

Reference

Takano, Mike. *Stochastic Oscillator, Technical Analysis of Stocks and Commodities*. April 1991.

Stein, John. *The Traders' Guide to Technical Indicators, Futures Magazine*. August 1990.

FastKCustomEasy (Function)

Disclaimer

The FastKCustomEasy function returns the Fast K line value in the Stochastic indicator using a single price input.

Syntax

```
FastKCustomEasy(Price,Length)
```

Returns (Double)

A numeric value containing the FastK line (using FastKCustom)..

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to consider for the stochastic calculations.
Length	Numeric	Sets the number of bars to consider.

Remarks

Please refer to the discussion under the Stochastic function.

Example

Assigns to Value1 the Stochastic FastK for prices over 14 bars.

```
Value1 = FastKCustomEasy(Close,14);
```

FastKCustomOrig (Function)

Disclaimer

The FastKCustomOrig function returns the Fast K value for the Stochastic oscillator based on user-defined price inputs.

Syntax

```
FastKCustomOrig(PriceH, PriceL, PriceC, StochLength)
```

Returns (Double)

A numeric value containing the FastKCustomOrig for the current bar.

Parameters

Name	Type	Description
PriceH	Numeric	Specifies which bar value (price, function, or formula) to use for the high in stochastic calculations.
PriceL	Numeric	Specifies which bar value (price, function, or formula) to use for the low in stochastic calculations.
PriceC	Numeric	Specifies which bar value (price, function, or formula) to use for the close in stochastic calculations.
StochLength	Numeric	Sets the number of bars to consider.

Remarks

This function differs from FastKCustom by using the original smoothing method suggested by George Lane.

Please refer to the discussion under the Stochastic function.

Example

Assigns to Value1 the Stochastic FastK for offset high and low prices over 14 bars.

```
Value1 = FastKCustomOrig(High+1,Low-1,Close,14);
```

Reference

Takano, Mike. *Stochastic Oscillator, Technical Analysis of Stocks and Commodities*. April 1991.

Stein, John. *The Traders' Guide to Technical Indicators, Futures Magazine*. August 1990.

FastLowestBar (Function)

Disclaimer

The FastLowestBar function uses a fast calculation method to return the number of bars ago the lowest price occurred. There may be times when two or more bars have exactly the same lowest value; when this happens the function identifies the most recent occurrence.

Syntax

```
FastLowestBar(Price, Length)
```

Returns (Integer)

A numeric value containing the number of bars ago that the lowest value occurred during the specified range of bars.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to compare for highest and lowest extremes.
Length	Numeric	Sets the number of bars to consider.

Remarks

The input Price can be hard coded with a bar attribute such as Close, Open, High, Low, and Volume or a numeric series type input. It can also be replaced with a valid EasyLanguage expression. For example: Close + Open, or Average(RSI(Close,14),14).

Note This function uses a fast calculation method that uses more memory than the traditional method.

Example

The following assigns the number of bars ago the lowest value of the last 14 bars occurred to Value1:

```
Value1 = FastLowestBar(Low, 14);
```

FindBar (Function)

Disclaimer

The FindBar function searches back in time for the first bar matching the date and time specified through the inputs TargetDate and TargetTime.

Syntax

```
FindBar(TargetDate, TargetTime)
```

Returns (Integer)

The bar number on which the target time and date occurs.

Parameters

Name	Type	Description
TargetDate	Numeric	Sets the date of the bar to find, entered in YYYMMDD format. Enter 1060115 for Jan. 15, 2006.
TargetTime	Numeric	Sets the time of the bar to find, entered in 24-hour military format. Enter 1300 for 1:00pm.

Remarks

It searches back through the full range of bars specified in the Maximum number of bars referenced by a study setting (known as MaxBarsBack). By default, FindBar requests 50 bars of data. If additional bars are needed to perform a calculation, the MaxBarsBack setting must be increased manually.

Example

Sets Value1 to the bar number found with the date of 5/10/2005 at 10:30am.

```
Value1 = FindBar(1050510, 1030);
```

.

FirstSession (Function)

Disclaimer

The `FirstSession` function returns the session number of the first session of a specified day.

Syntax

```
FirstSession(SessionType,XDay);
```

Returns (Integer)

A numeric value for the current bar.

Parameters

Name	Type	Description
`SessionType`	Numeric	Sets the type of session to reference. 0 = Auto Detect, 1 = Regular Session.
`XDay`	Numeric	Sets the day of the week that should be evaluated. 0=Sunday, 1=Monday, etc.

Remarks

The input parameter `SessionType` specifies the type of session information that should be returned. The type parameter may be specified as follows: Auto-Detect Session – [0, AutoSession] - uses whatever session is specified for the data series; regular or custom, or Regular Session – [1, RegularSession] - regular session information should always be returned. The input parameter `XDay` specifies the day of the week that should be evaluated for the Last session number. 0 = Sunday, 1 = Monday, 2 = Tuesday, etc.

Examples

Assigns to `Value1` the first session number for Wednesday:

```
Value1 = FirstSession(0,3);
```

See Also

FirstSessionMS

FirstSessionMS (Function)

Disclaimer

The FirstSessionMS function returns the session number of the first session of the specified day, based on the merged sessions in a multi-data chart.

MS - Merged sessions are from the earliest start time of all symbols to the latest end time for all sessions each trading day.

Syntax

```
FirstSessionMS(XDay);
```

Returns (Integer)

A numeric value for the current bar.

Parameters

Name	Type	Description
XDay	Numeric	Sets the day of the week that should be evaluated. 0=Sunday, 1=Monday, etc.

Examples

Assigns to Value1 the first session number of all merged sessions for Wednesday:

```
Value1 = FirstSessionMS(3);
```

See Also

FirstSession

Fisher (Function)

Disclaimer

The Fisher function calculates the Fisher transformation of a specified decimal value.

Syntax

```
Fisher(Price)
```

Returns (Double)

A numeric value containing the Fisher transformation value for `Price`. Returns a value of -999 if an error occurs.

Parameters

Name	Type	Description
`Price`	Numeric	Sets a decimal (>-1 and <1) value upon which to calculate the Fisher transformation.

Usage

This transformation produces a value that is, approximately, normally distributed rather than skewed. Use this function to perform hypothesis testing on the correlation coefficient. It will accept values greater than -1 and smaller than 1.

Example

The following expressions will calculate the Fisher transformation based on the correlation coefficient between two data streams:

```
Value1 = Correlation(Close of Data1, Close of Data2);
Value2 = Fisher(Value1);
```

FisherINV (Function)

Disclaimer

The `FisherINV` function calculates the inverse of the Fisher transformation.

Syntax

```
FisherINV(Price)
```

Returns (Double)

A numeric value containing the inverse Fisher transformation for the specified number.

Parameters

Name	Type	Description
`Price`	Numeric	Sets a number use to for calculating the inverse Fisher value.

Remarks

Use this transformation when analyzing correlations between ranges or arrays of data. If y = FISHER(x), then FISHERINV(y) = x.

Example

Assigns to `Value1` the Fisher inverse value based on an input of .97295.

```
Value1 = FisherINV(.97295);
```

Function Not Found (Function)

The function for which you requested information is either:

- Not included with TradeStation but rather created by a user or third party. For a listing of the functions included in TradeStation, refer to the Function Library.

or

- Not an EasyLanguage Reserved Word included with TradeStation. It is most likely a variable or input that you declared, a plot name, or part of a text string. For a listing of the Reserved Words provided by TradeStation Technologies, Inc., see the Reserved Word Library.

FundBoolean (Function)

Disclaimer

The FundBoolean function returns the boolean (true/false) value of the specified fundamental data from some number of periods ago.

Syntax

```
FundBoolean(FundFieldName,PeriodsAgo,oErrorCode)
```

Returns (Integer)

The function returns the true/false value of a specified fundamental data field from some number of periods ago based on a call to the reserved word GetFundAsBoolean. The oErrorCode output parameter returns a status code for the fundamental data request that should be checked after each use of FundBoolean to verify that the data was retrieved without error.

Parameters

Name	Type	Description
FundFieldName	String	Specifies the short name of the fundamental data to read.
PeriodsAgo	Numeric	Sets the number of periods ago from which to read the fundamental data.
oErrorCode	Numeric	Outputs the status of the fundamental data request. If no error occurred a constant value of fdrOK is returned, otherwise a specific error code is set. See reserved word GetLastFundDataError for a list of error codes.

Remarks

Fundamental data names are listed in tables under the following categories:

Snapshot, Balance Sheet Assets & Liabilities, Cash Flow, Income, Shareholder Equity

Note All units, except share values, are converted to Millions for all periods. Percentages are returned as whole values (i.e. a value of 26.5 represents 26.5% and not 2650.00%)

Typically, PeriodsAgo refers to a number of calendar quarters ago, since most companies report fundamental information (such as revenues and earnings) on a quarterly basis.

You can also access fundamental data from other data streams using by adding an alias after the function, such as 'of Data2'

See About Fundamental Data in TradeStation for more information about using the fundamental data as part of your fundamental analysis.

See Multiple Output Function for more information on using output parameters to return values.

Example

Assigns Value1 the boolean (true/false) value of fundamental data "Option" (Optionable stock) from two periods ago. The status of the request is returned using the output parameter RetError.

```
vars: RetError(-1);
Value1 = FundBoolean("Option", 2, RetError);
```

See Also

FundBoolean, FundDate, FundString

FundDate (Function)

Disclaimer

The FundDate function returns the date of the specified fundamental data from some number of periods ago.

Syntax

```
FundDate(FundFieldName,PeriodsAgo,oErrorCode)
```

Returns (Integer)

The function returns the posting date (in Julian date format) of a specified fundamental data field from some number of periods ago based on a call to the reserved word GetFundPostDate. The oErrorCode output parameter returns a status code for the fundamental data request that should be checked after each use of FundDate to verify that the data was retrieved without error.

Parameters

Name	Type	Description
FundFieldName	String	Specifies the short name of the fundamental data to read.
PeriodsAgo	Numeric	Sets the number of periods ago from which to read the fundamental data.
oErrorCode	Numeric	Outputs the status of the fundamental data request. If no error occurred a constant value of fdrOK is returned, otherwise a specific error code is set. See reserved word GetLastFundDataError for a list of error codes.

Remarks

Fundamental data names are listed in tables under the following categories:

Snapshot, Balance Sheet Assets & Liabilities, Cash Flow, Income, Shareholder Equity

Note All units, except share values, are converted to Millions for all periods. Percentages are returned as whole values (i.e. a value of 26.5 represents 26.5% and not 2650.00%)

Typically, PeriodsAgo refers to a number of calendar quarters ago, since most companies report fundamental information (such as revenues and earnings) on a quarterly basis.

You can also access fundamental data from other data streams using by adding an alias after the function, such as 'of Data2'

See About Fundamental Data in TradeStation for more information about using the fundamental data as part of your fundamental analysis.

See Multiple Output Function for more information on using output parameters to return values.

Example

Assigns Value1 the date value of fundamental data "ONET" (Net Income) from one period ago. The status of the request is returned using the output parameter RetError.

```
vars: RetError(-1);
Value1 = FundDate("ONET", 1, RetError);
```

See Also

FundValue, FundBoolean, FundString

FundPeriodEndDate (Function)

Disclaimer

The `FundPeriodEndDate` function returns the end date of the reporting period for the specified fundamental data from some number of periods ago.

Syntax

```
FundPeriodEndDate(FundFieldName,PeriodsAgo,oErrorCode)
```

Returns (Integer)

The function returns the end date of the reporting period (in Julian date format) of a specified fundamental data field from some number of periods ago. The `oErrorCode` output parameter returns a status code for the fundamental data request that should be checked after each use of `FundPeriodEndDate` to verify that the data was retrieved without error.

Parameters

Name	Type	Description
`FundFieldName`	String	Specifies the short name of the fundamental data to read.
`PeriodsAgo`	Numeric	Sets the number of periods ago from which to read the fundamental data.
`oErrorCode`	Numeric	Outputs the status of the fundamental data request. If no error occurred a constant value of fdrOK is returned, otherwise a specific error code is set. See reserved word GetLastFundDataError for a list of error codes.

Remarks

Fundamental data names are listed in tables under the following categories:

Snapshot, Balance Sheet Assets & Liabilities, Cash Flow, Income, Shareholder Equity

Note All units, except share values, are converted to Millions for all periods. Percentages are returned as whole values (i.e. a value of 26.5 represents 26.5% and not 2650.00%)

Typically, `PeriodsAgo` refers to a number of calendar quarters ago, since most companies report fundamental information (such as revenues and earnings) on a quarterly basis.

You can also access fundamental data from other data streams using by adding an alias after the function, such as 'of Data2'

See Multiple Output Function for more information on using output parameters to return values.

Example

Assigns `Value1` the end date value of fundamental data "ONET" (Net Income) from one period ago. The status of the request is returned using the output parameter RetError.

```
vars: RetError(-1);
Value1 = FundPeriodEndDate("ONET", 1, RetError);
```

See Also

FundValue, FundDate, FundBoolean, FundString

FundSetup (Function)

Disclaimer

The `FundSetup` returns a boolean value based on whether the value of a fundamental field meets certain momentum or acceleration-based conditions.

Syntax

```
FundSetup(FundField,MomOrAccel_0or1)
```

Returns (Integer)

The function returns a true value if the momentum or acceleration setup conditions are met for a specified fundamental data field, otherwise false.

Parameters

Name	Type	Description
`FundField`	String	Specifies the short name of the fundamental data to read.
`MomOrAccel_0or1`	Numeric	Sets whether to base the setup conditions on momentum or acceleration. 0 = momentum, 1 = acceleration

Remarks

Fundamentally-based setup conditions are based on the momentum or acceleration Momentum can be used to evaluate earnings (or other fundamental field) persistence and positive momentum. Acceleration can be used to evaluate positive acceleration of the fundamental field's value

Fundamental data names are listed in tables under the following categories:

Snapshot, Balance Sheet Assets & Liabilities, Cash Flow, Income, Shareholder Equity

Note All units, except share values, are converted to Millions for all periods. Percentages are returned as whole values (i.e. a value of 26.5 represents 26.5% and not 2650.00%)

See About Fundamental Data in TradeStation for more information about using the fundamental data as part of your fundamental analysis.

Example

Assigns `Value1` a true value if the momentum setup conditions for "RNTS" (Net Sales) are positive and false if momentum conditions are negative..

```
Value1 = FundSetup("RNTS", 0);
```

FundString (Function)

Disclaimer

The FundString function returns the string (text) value of the specified fundamental data from some number of periods ago.

Syntax

```
FundString(FundFieldName,PeriodsAgo,oErrorCode)
```

Returns (Integer)

The function returns the string value of a specified fundamental data field from some number of periods ago based on a call to the reserved word GetFundDataAsString. The oErrorCode output parameter returns a status code for the fundamental data request that should be checked after each use of FundString to verify that the data was retrieved without error.

Parameters

Name	Type	Description
FundFieldName	String	Specifies the short name of the fundamental data to read.
PeriodsAgo	Numeric	Sets the number of periods ago from which to read the fundamental data.
oErrorCode	Numeric	Outputs the status of the fundamental data request. If no error occurred a constant value of fdrOK is returned, otherwise a specific error code is set. See reserved word GetLastFundDataError for a list of error codes.

Remarks

Fundamental data names are listed in tables under the following categories:

Snapshot, Balance Sheet Assets & Liabilities, Cash Flow, Income, Shareholder Equity

Note All units, except share values, are converted to Millions for all periods. Percentages are returned as whole values (i.e. a value of 26.5 represents 26.5% and not 2650.00%)

Typically, PeriodsAgo refers to a number of calendar quarters ago, since most companies report fundamental information (such as revenues and earnings) on a quarterly basis.

You can also access fundamental data from other data streams using by adding an alias after the function, such as 'of Data2'

See About Fundamental Data in TradeStation for more information about using the fundamental data as part of your fundamental analysis.

See Multiple Output Function for more information on using output parameters to return values.

Example

Assigns Value1 the string value of fundamental data F_MGIND ('Industry Name') from two periods ago. The status of the request is returned using the output parameter RetError.

```
vars: RetError(-1);

Value1 = FundString("F_MGIND", 2, RetError);
```

See Also

FundValue, FundBoolean, FundDate

FundValue (Function)

Disclaimer

The FundValue function returns the numeric value of the specified fundamental data from some number of periods ago.

Syntax

```
FundValue(FundFieldName,PeriodsAgo,oErrorCode)
```

Returns (Integer)

The function returns the numeric value of a specified fundamental data field from some number of periods ago based on a call to the reserved word GetFundData. The oErrorCode output parameter returns a status code for the fundamental data request that should be checked after each use of FundValue to verify that the data was retrieved without error.

Parameters

Name	Type	Description
FundFieldName	String	Specifies the short name of the fundamental data to read.
PeriodsAgo	Numeric	Sets the number of periods ago from which to read the fundamental data.
oErrorCode	Numeric	Outputs the status of the fundamental data request. If no error occurred a constant value of fdrOK is returned, otherwise a specific error code is set. See reserved word GetLastFundDataError for a list of error codes.

Remarks

Fundamental data names are listed in tables under the following categories:

Snapshot, Balance Sheet Assets & Liabilities, Cash Flow, Income, Shareholder Equity

Note All units, except share values, are converted to Millions for all periods. Percentages are returned as whole values (i.e. a value of 26.5 represents 26.5% and not 2650.00%)

Typically, PeriodsAgo refers to a number of calendar quarters ago, since most companies report fundamental information (such as revenues and earnings) on a quarterly basis.

You can also access fundamental data from other data streams using by adding an alias after the function, such as 'of Data2'

See About Fundamental Data in TradeStation for more information about using the fundamental data as part of your fundamental analysis.

See Multiple Output Function for more information on using output parameters to return values.

Example

Assigns Value1 the numeric value of fundamental data "ONET" (Net Income) from one period ago. The status of the request is returned using the output parameter RetError.

```
vars: RetError(-1);
Value1 = FundValue("ONET", 1, RetError);
```

Assigns Value2 the numeric value of fundamental data "RNTS" (Net Sales) from three periods ago from an alternate data stream (Data2). The status of the request is returned using the output parameter RetError.

```
vars: RetError(-1);
Value2 = FundValue("RNTS", 3, RetError) of Data2;
```

See Also

FundBoolean, FundDate, FundString

FundValueTTM (Function)

Disclaimer

The FundValueTTM calculates the trailing twelve month (TTM) sum of a user-selected fundamental field.

Syntax

```
FundValueTTM(FundField,oErrorCode,oNewData)
```

Returns (Integer)

The function returns the sum over the training twelve month (TTM) period of a specified fundamental data field . The oErrorCode output parameter returns a status code for the fundamental data request that should be checked after each use of FundValueTTM to verify that the data was retrieved without error. The output parameter oNewData returns true is the sum includes data not on the immediately preceding bar.

Parameters

Name	Type	Description
FundFieldName	String	Specifies the short name of the fundamental data to read.
oErrorCode	Numeric	Outputs the status of the fundamental data request. If no error occurred a constant value of fdrOK is returned, otherwise a specific error code is set. See reserved word GetLastFundDataError for a list of error codes.
oNewData	Boolean	Outputs true if the TTM sum includes fundamental data that was not available on the immediately preceding bar, otherwise false.

Remarks

Fundamental data names are listed in tables under the following categories:

Snapshot, Balance Sheet Assets & Liabilities, Cash Flow, Income, Shareholder Equity

Note All units, except share values, are converted to Millions for all periods. Percentages are returned as whole values (i.e. a value of 26.5 represents 26.5% and not 2650.00%)

See About Fundamental Data in TradeStation for more information about using the fundamental data as part of your fundamental analysis.

See Multiple Output Function for more information on using output parameters to return values.

Example

Assigns Value1 the numeric value of trailing twelve month sum of fundamental data "ONET" (Net Income). The status of the request is returned using the output parameter RetError.

```
vars: RetError(-1), NewDataTF(false);
Value1 = FundValueTTM("ONET", RetError, NewDataTF);
```

See Also

FundValue

GCD (Function)

Disclaimer

The GCD function returns the greatest common denominator (GCD) of two numbers.

Syntax

```
GCD(Num1, Num2)
```

Returns (Double)

The function returns the GCD of two specified numbers.

Parameters

Name	Type	Description
`Num1`	Numeric	Sets the value of the first number to evaluate.
`Num2`	Numeric	Sets the value of the second number to evaluate.

Remarks

The GCD of more than two numbers can be calculated by finding the GCD two of the numbers, then find the GCD of that value and another number, and repeating the process as many times as necessary.

For example, the following formula finds the GCD of 3 numbers:

GCDxyz = GCD(GCD(x, y), z)

Examples

Displays GCD value of 0.00625 based on the input numbers .003125 and .01. The print statement formats the output to display 2 integer digits and 6 decimal digits.

```
Print(GCD(.003125, .01):2:6);
```

GenerateStrike (Function)

Disclaimer

The GenerateStrike function returns the strike price of an option based on the specified price proximity of the option to its underlying asset and the specified strike increment.

Syntax

```
GenerateStrike(ProximityToStrike, StrikeIncrement)
```

Returns (Double)

A numeric value representing the strike price of the option based on the number of strike above and below the current price of the underlying asset.

Parameters

Name	Type	Description
ProximityToStrike	Numeric	Sets the number of strike prices above or below the current price of the underlying asset that is to be returned by the function. Enter a positive value for strikes above the current price of the underlying asset; and a nega¬tive value for strikes below the current price of the underlying asset.
StrikeIncrement	Numeric	Sets the strike increment of the option.

Example

Assigns to Value1 the calculated strike price of an option that is going to be 2 strikes below the current underlying asset price, and with a 2.5 strike increment.

```
Value1 = GenerateStrike (-2, 2.5);
```

GetRGBValues (Function)

Disclaimer

A function that is used to get the component Red, Green, and Blue colors from an EasyLanguage RGB color value.

Syntax

```
GetRGBValues(BigRGBValue, oRedValue, oGreenValue, oBlueValue)
```

Returns

Returns the red, green, and blue values that make up the color indicated by BigRGBValue. The separate red, green, and blue values are returned to the caller by reference.

Parameters

Name	Type	Description
BigRGBValue	Numeric	Output variable returns a value representing one of 16 million color values.
oRedValue	Numeric	Output variable returns the Red portion of an EasyLanguage RGB color.
oGreenValue	Numeric	Output variable returns the Green portion of an EasyLanguage RGB color.
oBlueValue	Numeric	Output variable returns the Blue portion of an EasyLanguage RGB color.

Example

Returns the values of Red=0, Green=204, and Blue=56 for an EasyLanguage RGB color value of 3722240:

```
Value1 = GetRGBValues(3722240, oRedValue, oGreenValue, oBlueValue);
```

HarmonicMean (Function)

Disclaimer

The HarmonicMean function calculates the harmonic mean of prices over a range of bars.

Syntax

```
HarmonicMean(Price, Length)
```

Returns (Double)

A numeric value containing the HarmonicMean of a data set.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Length	Numeric	Sets the number of bars to consider.

Remarks

The harmonic mean is the reciprocal of the arithmetic mean of reciprocals. The harmonic mean is always less than the geometric mean, which is always less than the arithmetic mean.

Example

The following EasyLanguage expression will perform an action when the fast harmonic mean is greater than the slow harmonic mean:

```
If HarmonicMean(Close, 10) > HarmonicMean(Close, 20) Then {ACTION} ;
```

HarmonicMeanArray

Disclaimer

The HarmonicMeanArray calculates the harmonic mean of a specified array.

Syntax

```
HarmonicMeanArray(PriceArray, Size)
```

Returns (Double)

A numeric value containing the HarmonicMean of an array. If the function performs an illegal operation, or the declared Size Input is smaller than the declared array size, the function will return a -1.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing values upon which the harmonic mean is calculated.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.

Remarks

The harmonic mean is the reciprocal of the arithmetic mean of reciprocals. The harmonic mean is always less than the geometric mean, which is always less than the arithmetic mean.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The HarmonicMeanArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Note This array function is similar to HarmonicMean except that you can build an array of values that does not necessarily include contiguous bar values.

Example

Assigns to Value1 the harmonic mean of the specified named array MyArray with 10 elements.

```
Array: MyArray[10](0);
{… add EL statements here to assign price values to array elements... }
Value1 = HarmonicMeanArray(MyArray,10);
```

HeapPush (Function)

Disclaimer

The HeapPush function performs intermediate calculation to determine if heap sifting should continue as part of the SortHeapArray function.

Syntax

```
HeapPush(MyArray, ColIndx3, Size, Order)
```

Returns (Boolean)

The HeapPush function itself returns True if heap sifting should continue.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing values upon which the sort is performed.
ColIndx3	Numeric	Sets starting data element in the array.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.
Order	Numeric	Sets the sort order. 1 = sort descending, -1 = sort ascending

Remarks

A "heap sort" may be better suited for sorting of single-dimensional arrays of more than 25 elements compared to a simple sort (that may be faster for arrays of 25 or fewer elements).

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The HeapPush function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

HeapPush2D (Function)

Disclaimer

The HeapPush2D function performs intermediate calculation to determine if heap sifting should continue as part of the SortHeap2DArray function.

Syntax

```
HeapPush2D(MyArray, ColIndx3, Size1, Size2, Order)
```

Returns (Boolean)

The HeapPush2D function itself returns True if heap sifting should continue.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing values upon which the sort is performed.
ColIndx3	Numeric	Sets starting column in the array.
Size1	Numeric	Sets the row of the array to evaluate.
Size2	Numeric	Sets number of columns in the array to sort, starting with ColIndx.
Order	Numeric	Sets the sort order. 1 = sort descending, -1 = sort ascending

Remarks

A "heap sort" may be better suited for sorting of arrays of more than 25 elements compared to a simple sort (that may be faster for arrays of 25 or fewer elements).

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The HeapPush2D function only works with two-dimensional arrays. All array-based referencing begins with array element 1.

HeapSift (Function)

Disclaimer

The HeapSift performs a "heap sort" on an array as part of the SortHeapArray function..

Syntax

```
HeapSift(MyArray, ColIndx1, Size, Order)
```

Returns (Boolean)

The order of values of the array specified in MyArray are changed as the result of running HeapSift. The HeapSift function itself returns True.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing values upon which the sort is performed.
ColIndx1	Numeric	Sets starting data element in the array.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.
Order	Numeric	Sets the sort order. 1 = sort descending, -1 = sort ascending

Remarks

A "heap sort" may be better suited for sorting of single-dimensional arrays of more than 25 elements compared to a simple sort (that may be faster for arrays of 25 or fewer elements).

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The HeapSift function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Performs an ascending sort of the user declared array..

```
Array: myArray[30](0);
{… (assign values to array) }
Value1 = HeapSift(myArray, 30,-1);
```

HeapSift2D (Function)

Disclaimer

The `HeapSift2D` performs a "heap sort" on a town dimensional array as part of the `SortHeap2DArray` function.

Syntax

```
HeapSift2D(MyArray, ColIndx3, Size1, Size2, Order)
```

Returns (Boolean)

The order of values of the array specified in `MyArray` are changed as the result of running `HeapSift2D`. The `HeapSift2D` function itself returns True.

Parameters

Name	Type	Description
`PriceArray`	Numeric Array	Specifies the name of a numeric array containing values upon which the sort is performed.
`ColIndx3`	Numeric	Sets starting column in the array.
`Size1`	Numeric	Sets the row of the array to evaluate.
`Size2`	Numeric	Sets number of columns in the array to sort, starting with ColIndx.
`Order`	Numeric	Sets the sort order. 1 = sort descending, -1 = sort ascending

Remarks

A "heap sort" may be better suited for sorting arrays of more than 25 elements compared to a simple sort (that may be faster for arrays of 25 or fewer elements).

The value for the `Size` input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The `HeapSift2D` function only works with two-dimensional arrays. All array-based referencing begins with array element 1.

HighD (Function)

Disclaimer

The HighD series function allows you to reference the daily high of a previous day in an intraday chart (minute or tick-based) or a daily chart. HighD is one of a family of functions that allows historical references across various data intervals.

Syntax

```
HighD(PeriodsAgo)
```

Returns (Double)

The daily high price from a specified number of days ago. If the PeriodsAgo parameter is out of range (> 50), or there is not enough data, the function will return –1.

Parameters

Name	Type	Description
PeriodsAgo	Numeric	Sets the number of days/periods back to reference a previous day's high price. (50 days back maximum) (0 = Today's current high)

Remarks

You must have enough intraday data in the chart in order to look back and reference any previous daily high. For example, if you want to look back at the high of 25 days ago on a 5-minute chart, you must have at least 26 full days of 5-minute bars in the chart.

The value for the PeriodsAgo input parameter should always be a whole number greater than or equal to 0, but less than 51. Setting PeriodsAgo to 0 returns today's current high.

Example

In order to place a short limit order at the High of the previous day you would write:

```
SellShort Next Bar at HighD(1) Limit;
```

See Also

LowD, CloseD, OpenD, HighW, HighM, and HighY.

Highest (Function)

Disclaimer

The `Highest` function returns the highest price over a range of bars.

Syntax

```
Highest(Price, Length)
```

Returns (Double)

The highest `Price` found over a range of `Length` bars.

Parameters

Name	Type	Description
`Price`	Numeric	Specifies which bar value (price, function, or formula) to evaluate.
`Length`	Numeric	Sets the number of bars to consider.

Remarks

The input parameter `Price` can be a bar value such as `Close`, `High`, `Low`, `Open`, or `Volume`. It can also be any mathematical calculation such as: (`High` + `Low`) / 2, or a numeric function such as `RSI`, `Stochastic`, or `ADX`.

Example

The following assigns the highest calculated `Close + Open` value over the last 20 bars to `Value1`:

```
Value1 = Highest(Close + Open, 20);
```

See Also

HighestFC, HighestArray, Lowest, Extremes, HighestBar

HighestArray

Disclaimer

The HighestArray function returns the highest value in the specified price array.

Syntax

```
HighestArray(PriceArray, Size)
```

Returns (Double)

The highest value found in an array of values.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing bar values to compare for the highest value.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.

Remarks

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The HighestArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

The following assigns the highest value of the 20 elements in the array to Value1:

```
Array: myArray[20](0);
```

{… add EL statements here to assign High values to array elements... }

```
Value1 = HighestArray(MyArray, 20);
```

See Also

LowestArray, ExtremesArray, Highest

HighestBar (Function)

Disclaimer

The HighestBar function returns the number of bars ago the highest price occurred. There may be times when two or more bars have exactly the same highest value; when this happens the function identifies the most recent occurrence.

Syntax

```
HighestBar(Price, Length)
```

Returns (Integer)

The number of bars ago the highest Price occurred.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to evaluate.
Length	Numeric	Sets the number of bars to consider.

Remarks

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as: (High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

Example

The following assigns the number of bars ago the highest value of the last 14 bars occurred to Value1:

```
Value1 = HighestBar(High, 14);
```

See Also

Highest, LowestBar, Extremes

HighestFC (Series Function)

Disclaimer

The HighestFC (Fast Calculation) series function returns the highest price value over a range of bars.

Syntax

```
HighestFC(Price, Length)
```

Returns (Double)

The highest Price found over a range of Length bars.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to evaluate.
Length	Numeric	Sets the number of bars to consider.

Remarks

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as: (High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

Note This series function returns exactly the same value as Highest except that it uses a fast calculation method that takes slightly more memory than the non-FC version.

Example

The following assigns the highest High value for the last 14 bars to Value1:

```
Value1 = HighestFC(High, 14);
```

See Also

Highest, LowestFC, ExtremesFC

HighM (Function)

Disclaimer

The HighM series function allows you to reference the monthly high of a previous month in an intraday chart (minute or tick-based) or a daily, weekly, or monthly chart. HighM is one of a family of functions that allows historical references across various data intervals.

Syntax

```
HighM(PeriodsAgo)
```

Returns (Double)

The monthly high price from a specified number of months ago. If the PeriodsAgo parameter is out of range (> 50), or there is not enough data, the function will return –1.

Parameters

Name	Type	Description
PeriodsAgo	Numeric	Sets the number of months back to reference a previous month's high price. (50 months back maximum) (0 = This month's current high)

Remarks

You must have enough intraday data in the chart in order to look back and reference any previous monthly high. For example, if you want to look back at the high of 11 months ago on a 60-minute chart, you must have at least 12 full months of 60-minute bars in the chart.

The value for the PeriodsAgo input parameter should always be a whole number greater than or equal to 0, but less than 51. Setting PeriodsAgo to 0 returns this month's current high.

Example

In order to place a sell short limit order at the High of the previous month you would write

```
SellShort Next Bar at HighM(1) Limit;
```

See Also

LowM, CloseM, OpenM, HighD, HighW and HighY.

HighW (Function)

Disclaimer

The HighW series function allows you to reference the weekly high of a previous week in an intraday chart (minute or tick-based) or a daily or weekly chart. HighW is one of a family of functions that allows historical references across various data intervals.

Syntax

```
HighW(PeriodsAgo)
```

Returns (Double)

The weekly high price from a specified number of weeks ago. If the PeriodsAgo parameter is out of range (> 50), or there is not enough data, the function will return –1.

Parameters

Name	Type	Description
PeriodsAgo	Numeric	Sets the number of weeks back to reference a previous week's high price. (50 weeks back maximum) (0 = This week's current high)

Remarks

You must have enough intraday data in the chart in order to look back and reference any previous weekly high. For example, if you want to look back at the high of 15 weeks ago on a 30-minute chart, you must have at least 16 full weeks of 30-minute bars in the chart.

The value for the PeriodsAgo input parameter should always be a whole number greater than or equal to 0, but less than 51. Setting PeriodsAgo to 0 returns this week's current high.

Example

In order to place a sell short limit order at the High of the previous week you would write

```
SellShort Next Bar at HighW(1) Limit;
```

See Also

LowW, CloseW, OpenW, HighD, HighM and HighY.

HighY (Function)

Disclaimer

The HighY series function allows you to reference the yearly high of a previous year in an intraday chart (minute or tick-based) or a daily or yearly chart. HighY is one of a family of functions that allows historical references across various data intervals.

Syntax

```
HighY(PeriodsAgo)
```

Returns (Double)

The yearly high price from a specified number of years ago. If the PeriodsAgo parameter is out of range (> 50), or there is not enough data, the function will return –1.

Parameters

Name	Type	Description
PeriodsAgo	Numeric	Sets the number of years back to reference a previous year’s high price. (50 years back maximum) (0 = This year's current high)

Remarks

You must have enough intraday data in the chart in order to look back and reference any previous yearly high. For example, if you want to look back at the high of 7 years ago on a daily chart, you must have at least 8 full years of daily bars in the chart.

The value for the PeriodsAgo input parameter should always be a whole number greater than or equal to 0, but less than 51. Setting PeriodsAgo to 0 returns this week’s current high.

Example

In order to place a sell short limit order at the High of the previous year you would write

```
SellShort Next Bar at HighY(1) Limit;
```

See Also

LowY, CloseY, OpenY, HighD, HighW and HighM.

HPI (Function)

Disclaimer

The HPI series function returns the Herrick Payoff Index that is used to analyze futures and options. It is a measure of the money flow in and out of the market to which it is applied.

Syntax

```
HPI(OneCent, SmFactor)
```

Returns (Double)

A numeric value containing HPI for the current bar.

Parameters

Name	Type	Description
OneCent	Numeric	Sets the contract value of a 1 cent move in the underlying asset.
SmFactor	Numeric	Sets a user-defined smoothing constant, typically a decimal value between 0 and 1.

Remarks

Open Interest is a requirement in the Herrick Payoff Index, therefore, this study is only applicable to daily data for futures data. The formula applied is as follows:

$$HPI = \frac{\left(Ky + \left(CV(M-My)\left[1 \pm \frac{2I}{G}\right] - Ky\right)S\right)}{100{,}000}$$

. . .where

Ky = Yesterday's HPI

S = User-entered smoothing factor

C = the value of a 1 cent move

V = Volume

I = The absolute value of today's open interest or yesterday's open interest, which ever is greater.

G = the greater of today's or yesterday's open interest.

M = High minus Low and the difference is divided by two.

My = Yesterday's High minus yesterday's Low and the difference is divided by two.

The formula has a minus sign below a plus sign. If M > My the plus sign is used, however, if M < My the minus sign is used.

When the Herrick Payoff Index function is called by any analysis technique, it will require two parameters from the user. The first parameter, OneCent is the value of a 1 cent move, the letter C in the equation. This value will depend on the security that is loaded. For example, if the security being loaded was cattle this value would be $400. If the security being loaded was soybeans this value would be $50.

The second parameter, SmFactor, is the user entered smoothing factor, the letter S in the equation. The SmFactor more or less corresponds to a moving-average time span. Higher values used for the smoothing factor tend to give more reliable results.

Example

```
Plot1(HPI(.25, .133));
```

IFF (Function)

Disclaimer

The IFF function is used to conditionally return one of two specified numeric values.

Syntax

```
IFF(Test, TrueVal, FalseVal)
```

Returns (Double)

The numeric value of TrueVal if Test is true and the numeric value of FalseVal if Test is false.

Parameters

Test	Specifies a conditional expression to check (such as Close > Open).
TrueVal	Sets a numeric value to return if Test expression is true.
FalseVal	Sets a numeric value to return if Test expression is false.

Remarks

By using the IFF function, you are able to evaluate one or more conditions in the Test input expression, returning one numeric value if Test is true, and returning another numeric value if Test is false.

Example

Assigns to Value1 the number 1 if Close>Open is true or the number -1 if Close>Open is false.

```
Value1  = IFF(Close>Open,1,-1);
```

Reference

The IFF function was developed by TradeStation Technologies, Inc.

IFFLogic (Function)

Disclaimer

The IFFLogic function is used to conditionally return one of two specified true/false expressions.

Syntax

```
IFFLogic(Test, TrueVal, FalseVal)
```

Returns (Boolean)

The true/false value of TrueVal if Test is true; the true/false value of FalseVal if Test is false.

Parameters

Name	Type	Expression
Test	TrueFalse	Specifies the conditional expression to check (such as Close > Open).
TrueVal	TrueFalse	Sets the True/False expression to evaluate if Test condition is true.
FalseVal	TrueFalse	Sets the True/False expression to evaluate if Test condition is false.

Remarks

The IFFLogic function enables you use a Test condition to determine whether to evaluate either a 'True' expression or 'False' expression.

This function is similar to the IFF function, however the second and third inputs are true/false expressions, not numeric values.

Example

Assigns to Value1 the true/false value of the expression Close>Close[1] if Close>Open is true or the true/false value of the expression Open<Open[1] if Close>Open is false.

```
   Value1 = IFFLogic(Close>Open, Close>Close[1], Open<Open[1]);
```

Reference

The IFFLogic function was developed by TradeStation Technologies, Inc.

IFFString (Function)

Disclaimer

The IFFString function is used to conditionally return one of two specified string values.

Syntax

```
IFFString(Test, TrueVal, FalseVal)
```

Returns (String)

The string from input TrueVal if Test is true; the string from input FalseVal if Test is false.

Parameters

Name	Type	Description
Test	TrueFalse	Specifies an expression to check (such as Close > Open).
TrueVal	String	Sets the string to return if Test expression is true.
FalseVal	String	Sets the string to return if Test expression is false.

Remarks

By using the IFFString function, you are able to evaluate one or more conditions in the Test input expression, returning one string value if Test is true, and returning another string value if Test is false.

This function is similar to the IFF function, however the second and third inputs are text strings, not numeric values.

Examples

```
Plot1(IFFString(Close>Open,"True","False");
```

Reference

The IFFString function was developed by TradeStation Technologies, Inc.

ImpliedVolatility (Function)

Disclaimer

The ImpliedVolatility function calculates the market implied volatility for the specified option.

Syntax

```
ImpliedVolatility(ExpMonth, ExpYear, StrikePr, Rate100, MktVal, PutCall, AssetPr);
```

Returns (Double)

A numeric value representing the market volatility of the specified option.

Parameters

Name	Type	Description
ExpMonth	Numeric	Sets the expiration month of the option from 1 to 12 (enter January as 1).
ExpYear	Numeric	Sets the expiration year of the option in EasyLanguage date format, YYY (enter 106 for 2006).
StrikePr	Numeric	Sets the strike price of the option.
Rate100	Numeric	Sets the short-term risk free interest rate, usually the 90-day T-Bill, as a percentage (enter 4.9 for 4.9%).
MktVal	Numeric	Sets the market value of the option.
PutCall	Numeric	Sets if it is a Put or Call option. Put or 2 = Puts; Call or 3 = Calls.
AssetPr	Numeric	Sets the price of underlying asset.

Example

Assigns to Value1 the market implied volatility of a December 2006 Call option with the short-term 90-day T-Bill at 4.9%.

```
Value1 = ImpliedVolatility(12, 106, 70, 4.9, 8.125, Call, 73.875);
```

Intrinsic (Function)

Disclaimer

The Intrinsic function calculates the difference between the option’s strike price and the price of the underlying asset, which determines how much an option is in-the-money.

Syntax

```
Intrinsic(Price, Length, Mean, SDev)
```

Returns (Double)

A numeric value representing the intrinsic value of an option.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to be considered for the underlying asset.
OptType	Numeric	Sets if it is a Put or Call option. Put or 2 = Puts; Call or 3 = Calls.
StrikePr	Numeric	Sets the strike price of the option.

Example

Assigns to Value1 the intrinsic value of a Call option.

```
Value1 = Intrinsic(Close, 3, GenerateStrike(2, 2.5));
```

KeltnerChannel (Function)

Disclaimer

The KeltnerChannel function calculates the Keltner Channel value.

Syntax

```
KeltnerChannel(Price, Length, NumATRs)
```

Returns (Double)

A numeric value containing the Keltner Channel value for a specified bar.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to average.
Length	Numeric	Sets the period of time (in bars) over which an average will be taken.
NumATRs	Numeric	Sets a multiplier to be used in the calculation. Enter a positive value for the upper band and negative value for the lower band.

Usage

Keltner Channel is a channel that is based on a multiple (NumATRs) of average true ranges above and below a moving average. KeltnerChannel is most commonly used with studies or strategies that measure or take advantage of market volatility. It is similar in concept to the BollingerBand function.

Example

Assigns to Value1 the upper band of a Keltner Channel that is based on the Close, a 10 bar average, and a factor of 2.5.

```
Value1 = KeltnerChannel(Close, 10, 2.5)
```

Assigns to Value2 the lower band of a Keltner Channel that is based on the Close, a 20 bar average, and a factor of 1.5.

```
Value2 = KeltnerChannel(Low, 20, -1.5)
```

Additional Example

If you wanted to create an analysis technique that would alert you when the Close of a bar is higher than the value of the upper Keltner Channel using the closing prices over the last 14 bars and NumATRs of 2.5, you could use the following syntax:

```
Plot1(KeltnerChannel(Close, 14, 2.5), "KChannel");

If Close > Plot1 Then
Alert("Close is above the upper Keltner Band");
```

Kurtosis (Function)

Disclaimer

The Kurtosis function calculates the Kurtosis of a data set, which is based on the size of the tails of a distribution curve.

Syntax

```
Kurtosis(Price, Length)
```

Returns (Double)

A numeric value containing the Kurtosis of the specified data set. If Length is less than 3, the function returns zero.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use for the distribution.
Length	Numeric	Sets the number of bars used to build the distribution.

Remarks

Kurtosis characterizes the relative peakedness or flatness of a distribution compared with the normal distribution. Positive Kurtosis indicates a relatively peaked distribution. Negative Kurtosis indicates a relatively flat distribution.

Example

Assigns to Values1 the Kurtosis of the distribution of the closing prices during the last 100 periods:

```
Value1 = Kurtosis(Close, 100);
```

KurtosisArray (Function)

Disclaimer

The KurtosisArray function calculates the Kurtosis of values in an array.

Syntax

```
KurtosisArray(PriceArray, Size)
```

Returns (Double)

A numeric value containing the Kurtosis factor of the specified array. If the function performs an illegal operation, or the declared arrays or Size is not greater than 3, the function will return a 0.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing values upon which the Kurtosis is calculated.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.

Remarks

Kurtosis characterizes the relative peakedness or flatness of a distribution compared with the normal distribution. Positive Kurtosis indicates a relatively peaked distribution. Negative Kurtosis indicates a relatively flat distribution.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The KurtosisArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Note This array function is similar to Kurtosis except that you can build an array of values that does not necessarily include contiguous bar values.

Example

Assigns to Value1 the Kurtosis factor of the specified named array MyArray with 40 elements.

```
Array: MyArray[40](0);
```

{… add EL statements here to assign price values to array elements... }

```
Value1 = KurtosisArray(MyArray,40);
```

KurtosisOpt (Function)

Disclaimer

The KurtosisOpt function calculates the Kurtosis of a data set for which all the data points are not available..

Syntax

```
KurtosisOpt(Price, Length, Mean, SDev)
```

Returns (Double)

A numeric value containing the Kurtosis of the specified data set. If Length is less than 3, the function returns zero.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use for the distribution.
Length	Numeric	Sets the number of bars used to build the distribution.
Mean	Numeric	Sets the average of the period on which KurtosisOpt is being calculated.
SDev	Numeric	Sets the standard deviation of the period on which KurtosisOpt is being calculated.

Remarks

This function should be used in situations where the average and the standard deviation are known and all the data points are not available. If all the data points are available, the Kurtosis function should be used.

Kurtosis characterizes the relative peakedness or flatness of a distribution compared with the normal distribution. Positive Kurtosis indicates a relatively peaked distribution. Negative Kurtosis indicates a relatively flat distribution.

Example

The following expression returns the Kurtosis of the distribution of the closing prices of the last 100 periods, where the average over the last 100 bars was 152.5 and the standard deviation was 5.125.

```
Value1 = KurtosisOpt(Close, 100, 152.50, 5.125);
```

LastBarOnChart (Function)

Disclaimer

The LastBarOnChart function is used to determine if the current bar being evaluated is the last bar on the chart.

Syntax

```
LastBarOnChart
```

Returns (Boolean)

True if LastBarOnChart is the last charted bar. False if not.

Parameters

None

Example

In order to play a wave (sound) file only in the last bar of the chart you can write:

```
If LastBarOnChart Then
 Condition1 = PlaySound("C:\window\ding.wav");
```

Note This function will return True for all bars on a chart with tick-based interval, which have the same date and time as the last bar of the chart.

LastCalcDate (Function)

Disclaimer

The `LastCalcDate` function returns the date for the last completed bar.

Syntax

```
LastCalcDate
```

Returns (Integer)

A numeric value containing the date of the last completed bar in YYYMMDD format.

Parameters

None

Remarks

For example, if the function returns 960203, it means the last bar was completed on 02/03/96, or February 3, 1996.

Example

```
Value1 = LastCalcDate;
```

See Also

LastCalcTime

LastCalcTime (Function)

Disclaimer

The `LastCalcTime` function returns the time of completion (`Close`) of the last bar, in 24-hour military format (HHMM).

Syntax

```
LastCalcTime
```

Returns (Integer)

A numeric value containing the time of the last completed bar in 24-hour (HHMM) format.

Parameters

None

Remarks

For example, if the function returns 1700, the last bar was completed, or closed, at 5:00pm.

Example

```
Value1 = LastCalcTime;
```

See Also

LastCalcDate

LastDayOfMonth (Function)

Disclaimer

The `LastDayOfMonth` function returns the last calendar day of a month.

Syntax

```
LastDayOfMonth(TargetMonth)
```

Returns (Integer)

A numeric value representing the last calendar day of the specified month.

Parameters

Name	Type	Description
`TargetMonth`	Numeric	Sets the number of the month to be evaluated.

Remarks

Returns 28, 30, or 31, depending on the month being evaluated. There is no consideration given for leap years.

Examples

`LastDayOfMonth(03)` returns 31, indicating that the 31st is the last calendar day in March.

`LastDayOfMonth(Month(Date))` returns the last calendar day of the month using the `Month` function to get the month number from the `Date` reserved word.

LastHour (Function)

Disclaimer

The LastHour function determines if the current time is within the last hour of the first trading session.

Syntax

```
LastHour
```

Returns (Boolean)

True if the current time is within the last hour of the first trading session. False if the time is not within the last hour of the trading session.

Parameters

None

Remarks

LastHour returns True if referring to an S&P futures data stream and the time is 3:30pm, EST:

```
Condition1 = LastHour;
```

LastHour returns False if referring to Intel (INTC) and the time is 12:34pm, EST:

```
Condition2 = LastHour;
```

Examples

The following PaintBar study paints any bar within the last hour of the first session:

```
If LastHour Then Begin
 Plot1(Low, "PBLow");
 Plot2(High, "PBHi");
End;
```

LastSession (Function)

Disclaimer

The LastSession function returns the session number of the last session of a specified day.

Syntax

```
LastSession(SessionType,XDay);
```

Returns (Integer)

A numeric value for the current bar.

Parameters

Name	Type	Description
SessionType	Numeric	Sets the type of session to reference. 0 = Auto Detect, 1 = Regular Session.
XDay	Numeric	Sets the day of the week that should be evaluated. 0=Sunday, 1=Monday, etc.

Remarks

The input parameter SessionType specifies the type of session information that should be returned. The type parameter may be specified as follows: Auto-Detect Session – [0, AutoSession] - uses whatever session is specified for the data series; regular or custom, or Regular Session – [1, RegularSession] - regular session information should always be returned. The input parameter XDay specifies the day of the week that should be evaluated for the Last session number. 0 = Sunday, 1 = Monday, 2 = Tuesday, etc.

Example

Assigns to Value1 the last session number for Wednesday:

```
Value1 = LastSession(0,3);
```

See Also

LastSessionMS

LastSessionMS (Function)

Disclaimer

The `LastSessionMS` function returns the session number of the last session of the specified day, based on the merged sessions in a multi-data chart.

MS - Merged sessions are from the earliest start time of all symbols to the latest end time for all sessions each trading day.

Syntax

```
LastSessionMS(XDay);
```

Returns (Integer)

A numeric value for the current bar.

Parameters

Name	Type	Description
XDay	Numeric	Sets the day of the week to be evaluated 0=Sunday, 1=Monday, etc.

Remarks

The input parameter `XDay` specifies the day of the week that should be evaluated for the Last session number. 0 = Sunday, 1 = Monday, 2 = Tuesday, etc.

Examples

Assigns to `Value1` the last session number of all merged sessions for Wednesday:

```
Value1 = LastSessionMS (3);
```

See Also

LastSession

Leader (Function)

Disclaimer

The `Leader` function compares the current and previous bars to determine if the mid-point of the current bar is greater than the previous `High` or less than the previous `Low`.

Syntax

```
Leader
```

Returns (Integer)

The function returns 1 if the mid-point of the current bar is greater than the previous `High` or less than the previous `Low`, otherwise it returns 0.

Parameters

None

Example

```
Value1 = Leader;
```

LimitIfTouchedOrder (Function)

Disclaimer

The `LimitIfTouchedOrder` function is used to configure and send a limit if touched order using the order entry macro **.PlaceOrder**. You can call this function directly from your own EasyLanguage code to simplify the formatting and generation of macro orders.

NOTE Care should be exercised when calling this function as it is intended to send live orders. Confirmations for macro-generated orders can be configured by using the File -> Preferences -> TradeStation Order Entry menu sequence.

Syntax

```
LimitOrder(Frequency,Account,Action,SymbolCategory,Symbol,Quantity,Duration,GTDDate,
IfTouched,LimitPrice)
```

Returns (Integer)

`LimitIfTouchedOrder` returns 1 if called on a "real-time" tick and if a trade is allowed based on the user input "Frequency". The function returns -1 (negative one) in other cases; for example, if the function is called on an historical bar. If there are errors in the order parameters a runtime error message will be generated and the order will not be placed.

Parameters

Name	Type	Description	Supported Values
`Frequency`	String	Sets the order frequency.	Once, OncePerBar, EndOfBar, Always
`Account`	String	Sets the TradeStation account to be used for this order.	*Account string*
`Action`	String	Sets the order action.	Buy, Sell, SellShort, BuyToCover, BuyToOpen, BuyToClose, SellToOpen, SellToClose
`SymbolCategory`	String	Sets the trading category for the symbol'.	Equity, Future, Forex, EquityOption, FutureOption
`Symbol`	String	Sets the symbol to be used for this order.	*Symbol string*
`Quantity`	Numeric	Sets the number of shares, contracts, lots, etc. to be placed for this order.	Number
`Duration`	String	Sets the order duration.	Day, Day+, GTC, GTC+, GTD, GTD+, IOC, FOK, OPG, 1Min, 3Min, 5Min
`GTDDate`	String	Sets the date, if appropriate, to be used with the specified duration.	*Date Format* (MM/DD/YY).
`IfTouched`	Numeric	Sets the IfTouched price to be used for this order.	Number
`LimitPrice`	Numeric	Sets the Limit price to be used for this order.	Number

Remarks

The `LimitIfTouchedOrder` function uses the `PlaceOrder` function to format the order parameters and call the order entry macro. The `PlaceOrder` function is only intended to be used by this and other designated TradeStation functions.

This function disables advanced order placement features (All or None, Buy on minus,Sell on plus, etc.).

See .PlaceOrder command for more information on using the order entry macro for placing orders.

Example

Places a sell limit order of 100 shares for MSFT at a limit price of 24.35 using the order entry macro .PlaceOrder. Value1 returns a 1 if the order is valid.

```
Value1 =
LimitOrder("Once","SIM15180","Sell","Equity","MSFT",100,"Day","",24.65,24.35);
```

See Also

LimitlOrder, MarketIfTouchOrder, MarketOrder, StopLimitOrder, StopMarketOrder, TrailingStopOrder

LimitOrder (Function)

Disclaimer

The LimitOrder function is used to configure and send a limit order using the order entry macro **.PlaceOrder**. You can call this function directly from your own EasyLanguage code to simplify the formatting and generation of macro orders.

NOTE Care should be exercised when calling this function as it is intended to send live orders. Confirmations for macro-generated orders can be configured by using the File -> Preferences -> TradeStation Order Entry menu sequence.

Syntax

```
LimitOrder(Frequency,Account,Action,SymbolCategory,Symbol,Quantity,Duration,GTDDate,
LimitPrice)
```

Returns (Integer)

LimitOrder returns 1 if called on a "real-time" tick and if a trade is allowed based on the user input "Frequency". The function returns -1 (negative one) in other cases; for example, if the function is called on an historical bar. If there are errors in the order parameters a runtime error message will be generated and the order will not be placed.

Parameters

String parameters must be enclosed in quotation marks. Numeric values should not be in quotes..

Name	Type	Description	Supported Values
Frequency	String	Sets the order frequency.	Once, OncePerBar, EndOfBar, Always
Account	String	Sets the TradeStation account to be used for this order.	*Account string*
Action	String	Sets the order action.	Buy, Sell, SellShort, BuyToCover, BuyToOpen, BuyToClose, SellToOpen, SellToClose
SymbolCategory	String	Sets the trading category for the symbol'.	Equity, Future, Forex, EquityOption, FutureOption
Symbol	String	Sets the symbol to be used for this order.	*Symbol string*
Quantity	Numeric	Sets the number of shares, contracts, lots, etc. to be placed for this order.	Number
Duration	String	Sets the order duration.	Day, Day+, GTC, GTC+, GTD, GTD+, IOC, FOK, OPG, 1Min, 3Min, 5Min
GTDDate	String	Sets the date, if appropriate, to be used with the specified duration.	*Date Format* (MM/DD/YY).
LimitPrice	Numeric	Sets the Limit price to be used for this order.	Number

Remarks

The LimitOrder function uses the PlaceOrder function to format the order parameters and call the order entry macro. The PlaceOrder function is only intended to be used by this and other designated TradeStation functions.

This function disables advanced order placement features (All or None, Buy on minus,Sell on plus, etc.).

See .PlaceOrder command for more information on using the command line macro for placing orders.

Example

Places a sell limit order of 100 shares for MSFT at a limit price of 24.35 using the order entry macro .PlaceOrder. Value1 returns a 1 if the order is valid.

```
Value1 = LimitOrder("Once","SIM15180","Sell","Equity","MSFT",100,"Day","",24.35);
```

See Also

LimitIfTouchedOrder, MarketIfTouchOrder, MarketOrder, StopLimitOrder, StopMarketOrder, TrailingStopOrder

LinearReg (Function)

Disclaimer

The LinearReg function calculates the slope and angle of a linear regression line and allows you identify the price where the projected line crosses a future (or past) bar position.

Syntax

```
LinearReg(Price, Length, TgtBar, oLRSlope, oLRAngle, oLRIntercept, oLRValue)
```

Returns (Integer)

The oLRSlope, OLRAngle, OLRIntercept, and oLRValueRaw output parameters returns the slope, angle, intercept, and regression value. The LinearReg function itself returns a value of 1.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use for the calculating the regression line.
Size	Numeric	Sets the number of bars to consider.
TgtPos	Numeric	Sets a target bar position in the future (or past). Use a negative integer for a future bar, a positive integer for a previous bar, and zero for the current bar.
OLRSlope	Numeric	Outputs the slope of the linear regression line.
OLRAngle	Numeric	Outputs the angle of the linear regression line in degrees.
OLRIntercept	Numeric	Outputs the value at which the linear regression line crosses the current bar position.
oLRValueRaw	Numeric	Outputs the regression value where the linear regression line crosses the TgtPos bar position.

Remarks

Linear Regression is a concept also known as the "least squares method" or "best fit." Linear Regression attempts to fit a straight line between a range of bar values in such a way that distance between each data point and the line is minimized.

The equation of a regression line is:

$$y = mx + b$$

In the equation *m* refers to the slope of the regression line, *b* refers to the constant intercept of the y-axis, *x* is the independent variable, and *y* is the dependent variable.

The input Price can be hard coded with a bar attribute such as Close, Open, High, Low, and Volume or a numeric series type input. It can also be replaced with a valid EasyLanguage expression. For example: Close + Open, or Average(RSI(Close,14),14).

See Multiple Output Function for more information on using output parameters to return values.

Example

Assigns to Value2 the slope of a linear regression line of the Close, Value3 the angle, Value4 the current bar intercept value, and Value5 the regression value 5 bars into the future. Value1 is assigned a value of 1:

```
Vars: oOLRSlope(0), oLRAngle(0), oLRIntercept(0), oLRValueRaw(0);

{… add EL statements here to assign High values to array elements... }

Value1 = LinearReg (Close, 20, -5, oLRSlope, oLRAngle, oLRIntercept, oLRValueRaw);

Value2 = oOLRSlope;

Value3 = oLRAngle;

Value4 = oLRIntercept;

Value5 = oLRValueRaw;
```

LinearRegAngle (Function)

Disclaimer

Linear Regression is a concept also known as the "least squares method" or "best fit." Linear Regression attempts to fit a straight line between several data points in such a way that distance between each data point and the line is minimized.

Function

`LinearRegAngle(Price, Length)`

Parameters

`Price`	Specifies the value of interest is to be used.
`Length`	Specifies the number of bars to consider in the regression calculation.

Returns

A numeric value containing the angle of the current regression line.

Usage

The input `Price` can be hard coded with a bar attribute such as `Close`, `Open`, `High`, `Low`, and `Volume` or a numeric series type input. It can also be replaced with a valid EasyLanguage expression. For example: `Close + Open`, or `Average(RSI(Close,14),14)`.

The input `Length` can be hard coded replaced with a numeric simple type input.

Reference

Linear Regression is a principle found in most statistical publications.

LinearRegAngleFC (Function)

Disclaimer

Linear Regression is a concept also known as the "least squares method" or "best fit." Linear Regression attempts to fit a straight line between several data points in such a way that distance between each data point and the line is minimized.

Function

`LinearRegAngleFC(Price, Length)`

Parameters

`Price`	Specifies which price of the asset to use.
Length	Indicates number of bars used in the calculation.

Returns

A numeric value containing the angle of linear regression from the current bar.

Usage

The input `Price` can be hard coded with a bar attribute such as `Close`, `Open`, `High`, `Low`, and `Volume` or a numeric series type input. It can also be replaced with a valid EasyLanguage expression. For example: `Close + Open`, `or Average(RSI(Close,14),14).`

The input `Length` can be hard coded replaced with numeric simple type input.

Note This function uses a fast calculation method that uses more memory than the traditional method.

Reference

Linear Regression is a principle found in most statistical publications.

LinearRegFC (Function)

Disclaimer

Linear Regression is a concept also known as the "least squares method" or "best fit." Linear Regression attempts to fit a straight line between several data points in such a way that distance between each data point and the line is minimized.

This function uses a fast calculation method to determine the slope and angle of a linear regression line, and also determines the value of the line on the current bar or a specified number of bars ago or projected bars into the future.

Function

```
LinearRegFC(Price, Length, TgtBar, oLRSlope, oLRAngle, oLRIntercept, oLRValue)
```

Parameters

`Price`	Specifies the value of interest is to be used.
`Length`	Specifies the number of bars to consider in the regression calculation.
`TgtBar`	Represents the number of bars into the future or back into the past, zero for the current bar.
`oLRSlope`	Variable that receives the slope of the linear regression line.
`oLRAngle`	Variable that receives the angle of the linear regression line in terms of degrees.
`oLRIntercept`	Variable that receives the regression value for x number of bars out into the future or x number of bars back in the past.
`oLRValue`	Variable that receives the regression value for x number of bars out into the future or x number of bars back in the past.

Returns

`LinearRegFC` always returns 1. It calculates the linear regression line and passes the slope, angle, intercept and associated value by reference back to the calling function or technique.

Usage

The equation of any line resembles the following:

y = mx + b

In the equation *m* refers to the slope of the regression line, *b* refers to the constant intercept of the y-axis, *x* is the independent variable, and *y* is the dependent variable. This function returns all values.

The input `Price` can be hard coded with a bar attribute such as `Close`, `Open`, `High`, `Low`, and `Volume` or a numeric series type input. It can also be replaced with a valid EasyLanguage expression. For example: `Close + Open, or Average(RSI(Close,14),14).`

The inputs `Length` and `TgtBar` can be hard coded replaced with numeric simple type inputs.

Note This function uses a fast calculation method that uses more memory than the traditional method.

Reference

Linear Regression is a principle found in most statistical publications.

LinearRegSlope (Function)

Disclaimer

Linear Regression is a concept also known as the "least squares method" or "best fit." Linear Regression attempts to fit a straight line between several data points in such a way that distance between each data point and the line is minimized.

Function

`LinearRegSlope(Price, Length)`

Parameters

`Price`	Specifies the value of interest is to be used.
`Length`	Specifies the number of bars to consider in the regression calculation.

Returns

A numeric value containing the slope of the current regression line.

Usage

The input `Price` can be hard coded with a bar attribute such as `Close`, `Open`, `High`, `Low`, and `Volume` or a numeric series type input. It can also be replaced with a valid EasyLanguage expression. For example: `Close + Open, or Average(RSI(Close,14),14).`

The input `Length` can be hard coded replaced with a numeric simple type input.

Reference

Linear Regression is a principle found in most statistical publications.

LinearRegSlopeFC (Function)

Disclaimer

Linear Regression is a concept also known as the "least squares method" or "best fit." Linear Regression attempts to fit a straight line between several data points in such a way that distance between each data point and the line is minimized.

Function

`LinearRegSlopeFC(Price, Length)`

Parameters

`Price`	Specifies which price of the asset is to be used.
`Length`	The number of trailing bars to consider.

Returns

A numeric value containing the slope of the current regression line.

Usage

The input `Price` can be hard coded with a bar attribute such as `Close`, `Open`, `High`, `Low`, and `Volume` or a numeric series type input. It can also be replaced with a valid EasyLanguage expression. For example: `Close + Open, or Average(RSI(Close,14),14).`

The input `Length` can be hard coded replaced with a numeric simple type input.

Note This function uses a fast calculation method that uses more memory than the traditional method.

Reference

Linear Regression is a principle found in most statistical publications.

LinearRegValue (Function)

Disclaimer

Linear Regression is a concept also known as the "least squares method" or "best fit." Linear Regression attempts to fit a straight line between several data points in such a way that distance between each data point and the line is minimized.

The `LinearRegValue` function projects, based on the current regression line, the regression values for *x* number of bars out into the future or *x* number of bars back in the past.

Function

```
LinearRegValue(Price, Length, TgtBar)
```

Parameters

`Price`	Specifies the value of interest is to be used.
`Length`	Specifies the number of bars to consider in the regression calculation.
`TgtBar`	Represents the number of bars into the future or back into the past, zero for the current bar.

Returns

A numeric value containing the current value of the specified regression line at `TgtBar`.

Usage

The input `Price` can be hard coded with a bar attribute such as `Close`, `Open`, `High`, `Low`, and `Volume` or a numeric series type input. It can also be replaced with a valid EasyLanguage expression. For example: `Close + Open, or Average(RSI(Close,14),14).`

The input `Length` can be hard coded replaced with a numeric simple type input.

`TgtBar` represents the number of bars into the future or back into the past. If `TgtBar` is a positive number, `LinearRegValue` will be for a bar in the past. If `TgtBar` is negative, the `LinearRegValue` will be for a bar in the future. `TgtBar` can be replaced with a numeric simple input.

LinearRegValueFC (Function)

Disclaimer

Linear Regression is a concept also known as the "least squares method" or "best fit." Linear Regression attempts to fit a straight line between several data points in such a way that distance between each data point and the line is minimized.

The LinearRegValue function projects, based on the current regression line, the regression values for *x* number of bars out into the future or *x* number of bars back in the past.

Function

```
LinearRegValueFC(Price, Length, TgtBar)
```

Parameters

Price	Specifies which price of the asset to use.
Length	Indicates number of bars used in the calculation.
TgtBar	Represents the number of bars into the future or back into the past.

Returns

A numeric value containing the current value of the specified regression line at TgtBar.

Usage

The input Price can be hard coded with a bar attribute such as Close, Open, High, Low, and Volume or a numeric series type input. It can also be replaced with a valid EasyLanguage expression. For example: Close + Open, or Average(RSI(Close,14),14).

The input Length can be hard coded replaced with a numeric simple type input.

TgtBar represents the number of bars into the future or back into the past. If TgtBar is a positive number, LinearRegValue will be for a bar in the past. If TgtBar is negative, the LinearRegValue will be for a bar in the future. TgtBar can be replaced with a numeric simple input.

Note This function uses a fast calculation method that uses more memory than the traditional method.

Reference

Linear Regression is a principle found in most statistical publications.

LinRegArray (Function)

Disclaimer

The LinearReg function calculates the slope and angle of a linear regression line from an array of values and allows you identify the value where the projected line crosses a future (or past) array index position.

Syntax

```
LinRegArray(PriceArray, Size, TgtPos, oLRSlope, oLRAngle, oLRIntercept, oLRValueRaw)
```

Returns (Integer)

The oLRSlope, OLRAngle, OLRIntercept, and oLRValueRaw output parameters returns the slope, angle, intercept, and regression value. The LinRegArray function itself returns a value of 1.

Parameters

Name	Type	Description
PriceArray	NumericArray	Specifies the numeric array of data elements to evaluate.
Size	Numeric	Sets the number of elements (size) in the array to consider.
TgtPos	Numeric	Sets the point in the past or future to use for projecting a regression value. Use a positive integer for a previous point and a negative integer for a future point.
OLRSlope	Numeric	Outputs the slope of the linear regression line.
OLRAngle	Numeric	Outputs the angle of the linear regression line in degrees.
OLRIntercept	Numeric	Outputs the intercept value at which the linear regression line will intersect the last array element.
oLRValueRaw	Numeric	Outputs the regression value for a point TgtPos elements in the past or projected into the future.

Remarks

Linear Regression is a concept also known as the "least squares method" or "best fit." Linear Regression attempts to fit a straight line between several data points in such a way that distance between each data point and the line is minimized.

The equation of any line resembles the following:

y = mx + b

In the equation *m* refers to the slope of the regression line, *b* refers to the constant intercept of the y-axis, *x* is the independent variable, and *y* is the dependent variable. This function returns all values.

See Multiple Output Function for more information on using output parameters to return values.

Example

Assigns to Value2 the slope of a linear regression line, Value3 the angle, and Value4 the intercept point, and Value5 the projected intercept point from values in myArray. Value1 is assigned a value of 1:

```
Array: myArray[20](0);
Vars: oOLRSlope(0), oLRAngle(0), oLRIntercept(0), oLRValueRaw(0);
```

{… add EL statements here to assign High values to array elements... }

```
Value1 = LinRegArray (myArray, 20, 5, oLRSlope, oLRAngle, oLRIntercept,
oLRValueRaw);
Value2 = oOLRSlope;
Value3 = oLRAngle;
Value4 = oLRIntercept;
Value5 = oLRValueRaw;
```

LinRegArray2 (Function)

Disclaimer

Linear Regression is a concept also known as the "least squares method" or "best fit." Linear Regression attempts to fit a straight line between several data points in such a way that distance between each data point and the line is minimized.

This function uses two arrays (one with independent values and the other with dependent values) and calculates the slope and angle of a linear regression line, and also determines the value of the line on the current bar or a specified number of bars ago or projected bars into the future.

Function

```
LinRegArray2(PriceArray, Size, TgtPos, oLRSlope, oLRAngle, oLRIntercept, oLRValueRaw)
```

Parameters

`IndepArray`	Specifies the numeric array of independent price points to evaluate.
`DepArray`	Specifies the numeric array of dependent price points to evaluate.
`Size`	Specifies the number of array elements to consider in the regression calculation
`TgtPos`	Represents the number of bars into the future or back into the past, zero for the current bar.
`OLRSlope`	Variable that receives the slope of the linear regression line.
`OLRAngle`	Variable that receives the angle of the linear regression line in terms of degrees.
`OLRIntercept`	Variable that receives the intercept value at which the linear regression line will intersect the y-axis..
`oLRValueRaw`	Variable that receives the regression value for x number of bars out into the future or x number of bars back in the past.

Returns

`LinRegArray2` always returns 1. It calculates the linear regression line and passes the slope, angle, intercept and associated value by reference back to the calling function or technique.

Usage

The equation of any line resembles the following:

y = mx + b

In the equation *m* refers to the slope of the regression line, *b* refers to the constant intercept of the y-axis, *x* is the independent variable, and *y* is the dependent variable. This function returns all values.

LinRegForecastArray (Function)

Disclaimer

Used to calculate the predicted y-value for a given x-value, based on the linear regression calculation of an array.

Note All array-based function calculations begin with array element 1.

Function

```
LinRegForecastArray(PriceArray, Size, TgtPos)
```

Parameters

`PriceArray`	A numeric array that calculations are based on.
`Size`	A numeric expression representing the number of elements in the original array that have been used.
`TgtPos`	a numeric expression representing the data point for which you want to predict a value

Returns

A numeric value containing the predicted y-value for a given x-value, based on the linear regression calculation.

LinRegInterceptArray (Function)

Disclaimer

Uses regression analysis to determine the point at which a line will intersect the y-axis based on array values.

Note All array-based function calculations begin with array element 1.

Function

```
LinRegInterceptArray(PriceArray, Size)
```

Parameters

PriceArray	A numeric array that calculations are based on.
`Size`	A numeric expression representing the number of elements in the original array that have been used.

Returns

A numeric value containing the point at which a line will intersect the y-axis, using an array.

LinRegSlopeArray (Function)

Disclaimer

Calculates the slope of the linear regression line based on the values of array elements.

Note All array-based function calculations begin with array element 1.

Function

```
LinRegSlopeArray(PriceArray, Size)
```

Parameters

PriceArray	A numeric array that calculations are based on.
`Size`	A numeric expression representing the number of elements in the original array that have been used.

Returns

Returns the slope of the linear regression line, using an array.

LowD (Function)

Disclaimer

The LowD series function allows you to reference the daily Low of a previous day in an intraday chart (minute or tick-based) or a daily chart. LowD is one of a family of functions that allows historical references across various data intervals.

Syntax

```
LowD(PeriodsAgo)
```

Returns (Double)

The daily Low price from *num* days ago. If the PeriodsAgo parameter is out of range (> 50), or there is not enough data, the function will return –1.

Parameters

Name	Type	Description
PeriodsAgo	Numeric	The number of days/periods back to reference a previous day's low price. (50 days back maximum) (0 = Today's current Low)

Remarks

You must have enough intraday data in the chart in order to look back and reference any previous daily low. For example, if you want to look back at the low of 25 days ago on a 5-minute chart, you must have at least 26 full days of 5-minute bars in the chart.

The value for the PeriodsAgo input parameter should always be a positive whole number greater than or equal to 0, but less than 51. Setting PeriodsAgo to 0 returns today's current Low.

Example

In order to place a buy limit order at the Low of the previous day you would write:

```
Buy Next Bar at LowD(1) Limit;
```

See Also

HighD, CloseD, OpenD, LowW, LowM, and LowY.

Lowest (Function)

Disclaimer

The Lowest function returns the lowest price over a range of bars.

Syntax

```
Lowest(Price, Length)
```

Returns (Double)

The lowest Price found over the past Length bars.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to evaluate.
Length	Numeric	Sets the number of bars to consider.

Remarks

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as: (High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

Example

The following assigns the lowest Close value for the last 20 bars to Value1:

```
Value1 = Lowest(Close, 20);
```

See Also

LowestFC, LowestBar, LowestArray, Highest, Extremes

LowestArray (Function)

Disclaimer

The `LowestArray` function returns the lowest value in the specified array.

Syntax

```
LowestArray(PriceArray, Size)
```

Returns (Double)

The lowest value found in an array of values.

Parameters

Name	Type	Description
`PriceArray`	Numeric Array	Specifies the name of a numeric array containing bar values to compare for the lowest value.
`Size`	Numeric	Sets the number of data elements (size) in the array to evaluate.

Remarks

The value for the `Size` input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The `LowestArray` function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

The following assigns the lowest value of the 20 elements in the array to `Value1`:

```
Array: myArray[20](0);
```

{… add EL statements here to assign `Low` values to array elements... }

```
Value1 = LowestArray(MyArray, 20);
```

See Also

HighestArray, ExtremesArray, Lowest

LowestBar (Function)

Disclaimer

The LowestBar function returns the number of bars ago the lowest price occurred. There may be times when two or more bars have exactly the same lowest value; when this happens the function identifies the most recent occurrence.

Syntax

```
LowestBar(Price, Length)
```

Returns (Integer)

The number of bars ago the lowest Price occurred.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to evaluate.
Length	Numeric	Sets the number of bars to consider.

Remarks

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as: (High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

Example

The following assigns the number of bars ago the lowest value of the last 14 bars occurred to Value1:

```
Value1 = LowestBar(Low, 14);
```

See Also

Lowest, HighestBar, Extremes

LowestFC (Series Function)

Disclaimer

The LowestFC (Fast Calculation) series function returns the lowest price value over a range of bars.

Syntax

```
LowestFC(Price, Length)
```

Returns (Double)

The lowest Price found over the past Length bars.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to evaluate.
Length	Numeric	Sets the number of bars to consider.

Remarks

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as: (High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

Note This series function returns exactly the same value as Lowest except that it uses a fast calculation method that takes slightly more memory than the non-FC version.

Example

The following assigns the highest High value for the last 14 bars to Value1:

```
Value1 = LowestFC(Low, 14);
```

See Also

Lowest, HighestFC, ExtremesFC

LowM (Function)

Disclaimer

The LowM series function allows you to reference the monthly Low of a previous month in an intraday chart (minute or tick-based) or a daily, weekly, or monthly chart. LowM is one of a family of functions that allows historical references across various data intervals.

Usage

```
LowM(PeriodsAgo)
```

Returns (Double)

The monthly Low price from *num* months ago. If the PeriodsAgo parameter is out of range (> 50), or there is not enough data, the function will return –1.

Parameters

Name	Type	Description
PeriodsAgo	NumericSimple	The number of months back to reference a previous month's Low price. (50 months back maximum) (0 = This month's current Low)

Remarks

You must have enough intraday data in the chart in order to look back and reference any previous monthly low. For example, if you want to look back at the low of 11 months ago on a 60-minute chart, you must have at least 12 full months of 60-minute bars in the chart.

The value for the PeriodsAgo input parameter should always be a positive whole number greater than or equal to 0, but less than 51. Setting PeriodsAgo to 0 returns this month's current Low.

Example

In order to place a buy limit order at the Low of the previous month you would write:

```
Buy Next Bar at LowM(1) Limit;
```

See Also

HighM, CloseM, OpenM, LowD, LowW, and LowY.

LowW (Function)

Disclaimer

The LowW series function allows you to reference the weekly Low of a previous week in an intraday chart (minute or tick-based) or a daily or weekly chart. LowW is one of a family of functions that allows historical references across various data intervals.

Usage

```
LowW(PeriodsAgo)
```

Returns (Double)

The weekly Low price from *num* weeks ago. If the PeriodsAgo parameter is out of range (> 50), or there is not enough data, the function will return –1.

Parameters

Name	Type	Description
PeriodsAgo	NumericSimple	The number of weeks back to reference a previous week's Low price. (50 weeks back maximum) (0 = This week's current Low)

Remarks

You must have enough intraday data in the chart in order to look back and reference any previous weekly low. For example, if you want to look back at the low of 15 weeks ago on a 30-minute chart, you must have at least 16 full weeks of 30-minute bars in the chart.

The value for the PeriodsAgo input parameter should always be a positive whole number greater than or equal to 0, but less than 51. Setting PeriodsAgo to 0 returns this week's current Low.

Example

In order to place a buy limit order at the Low of the previous week you would write:

```
Buy Next Bar at LowW(1) Limit;
```

See Also

HighW, CloseW, OpenW, LowD, LowM, and LowY.

LowY (Function)

Disclaimer

The LowY series function allows you to reference the yearly Low of a previous year in an intraday chart (minute or tick-based) or a daily, weekly, monthly, or yearly chart. LowY is one of a family of functions that allows historical references across various data intervals.

Usage

```
LowY(PeriodsAgo)
```

Returns (Double)

The yearly Low price from *num* years ago. If the PeriodsAgo parameter is out of range (> 50), or there is not enough data, the function will return –1.

Parameters

Name	Type	Description
PeriodsAgo	NumericSimple	The number of years back to reference a previous year's Low price. (50 years back maximum) (0 = This year's current Low)

Remarks

You must have enough intraday data in the chart in order to look back and reference any previous yearly low. For example, if you want to look back at the low of 5 years ago on a daily chart, you must have at least 6 full years of daily bars in the chart.

The value for the PeriodsAgo input parameter should always be a positive whole number greater than or equal to 0, but less than 51. Setting PeriodsAgo to 0 returns this year's current Low.

Example

In order to place a buy limit order at the Low of the previous year you would write:

```
Buy Next Bar at LowY(1) Limit;
```

See Also

HighY, CloseY, OpenY, LowD, LowW, and LowM.

LRO (Function)

Disclaimer

The LRO (Least Recent Occurence) function returns the number of bars ago the specified expression was True. Or, if the specified expression did not occur within the last *x* number of bars, the function informs you of such.

Syntax

```
LRO(Test, Length, Instance)
```

Returns (Integer)

A numeric value containing the number of bars ago that the specified Test was True.

Parameters

Name	Type	Description
Test	TrueFalse	Sets the true/false expression to check for (that is, Close > Open).
Length	Numeric	Sets the number of bars to check.
Instance	Numeric	Sets which occurrence, for example, 1 = most recent, 2 = 2nd most recent, and so on.

Remarks

The LRO function is specifically designed to identify when a certain condition occurred. It checks from the bar furthest away and works toward the current bar.

The function always returns a number representing how many bars ago the true/false expression was fulfilled (0 = current bar, 1 = one bar ago, 2 = 2 bars ago …). If the function does not find a bar within the specified Length, which meets the criteria, it will return a -1.

Note When using the LRO function in an analysis technique, always check the result of the LRO function before using the value in another calculation or function. If the condition occurred over Length number of bars, a -1 value will be returned.

Example

```
Value1 = LRO(Close>Open, 5, 1);
```

See Also

MRO

LWAccDis (Function)

Disclaimer

The `LWAccDis` series function accumulates price differences based on a method described by author Larry Williams.

Syntax

```
LWAccDis
```

Returns (Double)

A numeric value containing the accumulation distribution for the current bar.

Parameters

None

Example

```
Plot1(LWAccDis);
```

Reference

Williams, Larry. *Secret of Selecting Stocks*. Windsor Books. New York.

MACD (Function)

Disclaimer

The `MACD` (Moving Average Convergence Divergence) series function returns the difference between a fast and slow exponential moving average based on the same `Price`.

Syntax

```
MACD(Price, FastLength, SlowLength)
```

Returns (Double)

A numeric value containing `MACD` for the current bar.

Parameters

Name	Type	Description
`Price`	Numeric	Specifies which bar value (price, function, or formula) to be considered.
`FastLength`	Numeric	Sets the number of bars to consider for the fast average.
`SlowLength`	Numeric	Sets the number of bars to consider for the slow average

Remarks

`FastLength` and `SlowLength` refer to the number of bars used in the moving averages. This should be a whole number that cannot change on a bar-by-bar basis. `FastLength`, by definition, should be less than `SlowLength`. If the specified length of `FastLength` is greater than that of the `SlowLength`, the oscillator will invert.

Example

```
Plot1(MACD(Close,16,26);
```

Reference

Gerald Appel, Signalert Corporation, 150 Great Neck Road, Great Neck, NY 11021

MarketIfTouchedOrder (Function)

Disclaimer

The MarketIfTouchedOrder function is used to configure and send a limit if touched order using the order entry macro **.PlaceOrder**. You can call this function directly from your own EasyLanguage code to simplify the formatting and generation of macro orders.

NOTE Care should be exercised when calling this function as it is intended to send live orders. Confirmations for macro-generated orders can be configured by using the File -> Preferences -> TradeStation Order Entry menu sequence.

Syntax

```
MarketIfTouchedOrder(Frequency,Account,Action,SymbolCategory,Symbol,Quantity,Duration,GTDDate,IfTouched)
```

Returns (Integer)

MarketIfTouchedOrder returns 1 if called on a "real-time" tick and if a trade is allowed based on the user input "Frequency". The function returns -1 (negative one) in other cases; for example, if the function is called on an historical bar. If there are errors in the order parameters a runtime error message will be generated and the order will not be placed.

Parameters

Name	Type	Description	Supported Values
Frequency	String	Sets the order frequency.	Once, OncePerBar, EndOfBar, Always
Account	String	Sets the TradeStation account to be used for this order.	*Account string*
Action	String	Sets the order action.	Buy, Sell, SellShort, BuyToCover, BuyToOpen, BuyToClose, SellToOpen, SellToClose
SymbolCategory	String	Sets the trading category for the symbol'.	Equity, Future, Forex, EquityOption, FutureOption
Symbol	String	Sets the symbol to be used for this order.	*Symbol string*
Quantity	Numeric	Sets the number of shares, contracts, lots, etc. to be placed for this order.	Number
Duration	String	Sets the order duration.	Day, Day+, GTC, GTC+, GTD, GTD+, IOC, FOK, OPG, 1Min, 3Min, 5Min
GTDDate	String	Sets the date, if appropriate, to be used with the specified duration.	*Date Format* (MM/DD/YY).
IfTouched	Numeric	Sets the IfTouched price to be used for this order.	Number

Remarks

The MarketIfTouchedOrder function uses the PlaceOrder function to format the order parameters and call the order entry macro. The PlaceOrder function is only intended to be used by this and other designated TradeStation functions.

This function disables advanced order placement features (All or None, Buy on minus,Sell on plus, etc.).

See .PlaceOrder command for more information on using the order entry macro for placing orders.

Example

Places a sell limit order of 100 shares for MSFT at a limit price of 24.35 using the order entry macro .PlaceOrder. Value1 returns a 1 if the order is valid.

```
Value1 =
MarketIfTouchedOrder("Once","SIM15180","Sell","Equity","MSFT",100,"Day","",24.65);
```

See Also

LimitIfTouchedOrder, LimitIOrder, MarketOrder, StopLimitOrder, StopMarketOrder, TrailingStopOrder

MarketOrder (Function)

Disclaimer

The MarketOrder function is used to configure and send a limit order using the order entry macro **.PlaceOrder**. You can call this function directly from your own EasyLanguage code to simplify the formatting and generation of macro orders.

NOTE Care should be exercised when calling this function as it is intended to send live orders. Confirmations for macro-generated orders can be configured by using the File -> Preferences -> TradeStation Order Entry menu sequence.

Syntax

```
MarketOrder(Frequency,Account,Action,SymbolCategory,Symbol,Quantity)
```

Returns (Integer)

MarketOrder returns 1 if called on a "real-time" tick and if a trade is allowed based on the user input "Frequency". The function returns -1 (negative one) in other cases; for example, if the function is called on an historical bar. If there are errors in the order parameters a runtime error message will be generated and the order will not be placed.

Parameters

Name	Type	Description	Supported Values
Frequency	String	Sets the order frequency.	Once, OncePerBar, EndOfBar, Always
Account	String	Sets the TradeStation account to be used for this order.	*Account string*
Action	String	Sets the order action.	Buy, Sell, SellShort, BuyToCover, BuyToOpen, BuyToClose, SellToOpen, SellToClose
SymbolCategory	String	Sets the trading category for the symbol'.	Equity, Future, Forex, EquityOption, FutureOption
Symbol	String	Sets the symbol to be used for this order.	*Symbol string*
Quantity	Numeric	Sets the number of shares, contracts, lots, etc. to be placed for this order.	Number

Remarks

The MarketOrder function uses the PlaceOrder function to format the order parameters and call the order entry macro. The PlaceOrder function is only intended to be used by this and other designated TradeStation functions.

This function disables advanced order placement features (All or None, Buy on minus,Sell on plus, etc.).

See .PlaceOrder command for more information on using the command line macro for placing orders.

Example

Places a buy order of 100 shares for MSFT at market using the order entry macro .PlaceOrder. Value1 returns a 1 if the order is valid.

```
Value1 = MarketOrder("Once","SIM15180","Sell","Equity","MSFT",100);
```

See Also

LimitIfTouchedOrder, LimitlOrder, MarketIfTouchOrder, StopLimitOrder, StopMarketOrder, TrailingStopOrder

MassIndex (Function)

Disclaimer

The `MassIndex` series function returns a value that is used to warn of an impending direction change.

Syntax

```
MassIndex(SmoothingLength, SummationLength)
```

Returns (Double)

A numeric value containing `MassIndex` for the current bar.

Parameters

Name	Type	Description
`SmoothingLength`	Numeric	Sets the number of periods for the exponential moving average.
`SummationLength`	Numeric	Sets the number of periods for the summation and second exponential moving average.

Remarks

The formula for the function is:

`MassIndex` = Numerator/Denominator

…where

```
Numerator = Summation(XAverage(High-Low, Smoothing), Summation)
Denominator = XAverage(XAverage(High-Low, Smoothing), Summation)
```

Example

```
Plot1(MassIndex(9,25));
```

Reference

Dorsey, Donald. *The Mass Index. Technical Analysis of Stocks and Commodities*. June 1992, Pages 67-69.

McClellanOsc (Function)

Disclaimer

The McClellanOsc series function calculates the smoothed difference between the New York Stock Exchange Advancing and Declining Issues which is a measure of measure of market breadth .

Syntax

```
McClellanOsc(AdvIssues, DecIssues, FastLength, SlowLength)
```

Returns (Double)

A numeric value containing the McClellan Oscillator for the current bar.

Parameters

Name	Type	Description
AdvIssues	Numeric	Specifies which bar value (price, function, or formula) to be considered for the numeric series representing advancing Issues
DecIssues	Numeric	Specifies which bar value (price, function, or formula) to be considered for the numeric series representing declining Issues
FastLength	Numeric	Sets the number of bars used to calculate the exponentially smoothed average of the declining issues
SlowLength	Numeric	Set the number of bars used to calculate the exponentially smoothed average of the advancing issues

Remarks

McClellan Oscillator formations help to identify changes in market direction.

Examples

The McClellan Oscillator calculation is based on the Advancing Issues (plotted in Data1) and the Declining Issues (plotted in Data2). The Advancing issues (Data1) are smoothed by a 39 bar exponential average, while the Declining issues (Data2) are smoothed by a 19 bar exponential average.

```
Value1 = McClellanOsc(Close of Data1, Close of Data2, 19, 39);
```

If you wanted a mark on the High of a bar if the McClellan Oscillator became Overbought (greater than 100), you could use the following syntax:

```
If McClellanOsc(Close of Data1, Close of Data2, 19, 39) > 100 Then
 Plot1(High, "OverBght");
```

Median (Function)

Disclaimer

The Median function returns the middle value after sorting numbers over a specified range of bars.

Syntax

```
Median(Price, Length)
```

Returns (Double)

A numeric value containing the median value of a specified number of sorted values.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Length	Numeric	Sets the number of bars to consider.

Remarks

The input Price tells the function what values to look at, while the Length input tells the function how many bars to include in the calculations. The function will sort Length number of Price elements in ascending order and return the middle element. If Length is an even number, it will choose the two values that fall at the middle of the sort and average them. See Table 1. If Length is an odd number the function will return the middle value. See Table 2.

Table 1

Bar Number	Price	Rank (1 = smallest)	Median
Close[0]	3646.6	❷	
Close[1]	3625.0	❶	
Close[2]	3667.1	❸	
Close[3]	3669.6	❹	
	Length = 4		3656.8

Table 2

Bar Number	Price	Rank (1 = smallest)	Median
Close[0]	3646.6	❷	
Close[1]	3625.0	❶	
Close[2]	3667.1	❸	
Close[3]	3669.6	❹	
Close[4]	3670.0	❺	
	Length = 5		3667.1

Price is a numeric series type input. The input Length must be a whole number as it represents the number of bars, and it cannot change on a bar-to-bar basis.

Example

```
Value1 = Median(Close,10);
```

MedianArray (Function)

Disclaimer

The MedianArray function looks at an array of values and returns the median value.

Syntax

```
MedianArray(PriceArray, Size)
```

Returns (Double)

A numeric value containing the median value for the array.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing values upon which the median value is calculated.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.

Usage

If you had an array with 4 values, the function would sort the array in ascending order, choose the second and third largest values, and average them. See Table 1. If the array contained 5 values, the function would sort the array in ascending order, and choose the third (middle) value. See Table 2.

Table 1

ArrayElement	Value	Rank (1 = smallest)	Median
ArrayElement[1]	3646.6	❷	
ArrayElement[2]	3625.0	❶	
ArrayElement[3]	3667.1	❸	
ArrayElement[4]	3669.6	❹	
			3656.8

Table 2

ArrayElement	Value	Rank (1 = smallest)	Median
ArrayElement[1]	3646.6	❷	
ArrayElement[2]	3625.0	❶	
ArrayElement[3]	3667.1	❸	
ArrayElement[4]	3669.6	❹	
ArrayElement[5]	3670.0	❺	
			3667.1

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The MedianArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Note This array function is similar to Median except that you can build an array of values that does not necessarily include contiguous bar values.

Example

Assigns to Value1 the median value of the specified named array MyArray with 10 elements.

```
Array: MyArray[10](0);
{… add EL statements here to assign price values to array elements... }
Value1 = MedianArray(MyArray,10);
```

See Also

ExtremesArray, DevSqrdArray

MedianPrice (Function)

Disclaimer

The `MedianPrice` function returns the median (mid) price of a bar.

Syntax

```
MedianPrice
```

Returns (Double)

A numeric value containing the median bar price.

Parameters

None

Remarks

The median price is the sum of the `High` and `Low` of a bar divided by two or (H+L)/2.

Example

Assigns to Value1 the median price of 5 bars ago.

```
Value1 = MedianPrice[5];
```

MFI (Function)

Disclaimer

The MFI function returns the Market Facilitation Index value.

Syntax

```
MFI(AnyVol)
```

Returns (Double)

A numeric value containing MFI for the current bar.

Parameters

Name	Type	Description
AnyVol	Numeric	Sets the volume value (Volume or Ticks) to use

Remarks

The AnyVol input parameter should be set to Volume when referencing daily data and to Ticks when referencing tick/minute data.

The formula for MFI is:

```
MFI = Range / AnyVol
```

Please refer to the discussion on the Range.

Example

Plots MFI for intra-day or tick data.

```
Plot1(MFI(Ticks),"MFI");
```

MidPoint (Function)

Disclaimer

The `MidPoint` function finds the highest and lowest value of a `Price` over a given `Length` and returns the average of the two.

Syntax

```
MidPoint(Price, Length)
```

Returns (Double)

The `MidPoint` price of the period specified.

Parameters

Name	Type	Description
`Price`	Numeric	Specifies which bar value (price, function, or formula) to be use for the calculation.
`Length`	Numeric	Sets the number of bars to consider.

Remarks

The input parameter `Price` can be a bar value such as `Close`, `High`, `Low`, `Open`, or `Volume`. It can also be any mathematical calculation such as: (`High` + `Low`) / 2, or a numeric function such as `RSI`, `Stochastic`, or `ADX`.

Example

```
Plot1(MidPoint(C,10));
```

MinutesIntoWeek (Function)

Disclaimer

The `MinutesIntoWeek` function returns the number of minutes since 12 am Sunday.

Syntax

```
MinutesIntoWeek(XDay, XTime))
```

Returns (Integer)

The total minutes from 12 am Sunday to the specified XDay and XTime.

Parameters

Name	Type	Description
`XDay`	Numeric	Sets an integer value for the day of the week. Enter 0 for Sunday, 1 for Monday, through 6 for Saturday.
`XTime`	Numeric	Specifies the time in 24 hour HHMM format. For example, enter 2150 for 9:50 pm.

Example

Assigns to `Value1` the number of minutes between Sunday at 12:00 am and Monday at 10:20 am, which is 2060 minutes.

```
Value1 = MinutesIntoWeek(1, 1020);
```

MinutesToTime (Function)

Disclaimer

The MinutesToTime function converts a number of minutes from midnight to a 24-hour military time format (HHMM).

Syntax

```
MinutesToTime(Minutes)
```

Returns (Integer)

A numeric value containing the time in military format (for example, 1530 = 3:30pm).

Parameters

Name	Type	Description
Minutes	Numeric	Specifies the number of minutes since midnight

Example

Assigns to Value1 the 24-hour military time value of 0130 based on an input of 90 minutes.

```
Value1 = MinutesToTime(90);
```

Mode (Function)

Disclaimer

The Mode function returns the most frequently occurring or repetitive value over a specified range of bars.

Syntax

```
Mode(Price, Length, Type)
```

Returns (Double)

A numeric value containing the most frequently occurring or repetitive value in a specified period. If there is no repetition in the range of data, the Function will return a -1.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Length	Numeric	Sets the number of bars to consider.
Type	Numeric	A numeric value that determines if the higher (1) or lower (-1) of the values will be displayed in the event that there are multiple repetitious values in the data series.

Examples

Assigns to Value1 the most frequently occurring Close value in the last 21 bars. If there happens to be multiple sets of frequently occurring Close values, the higher valued of the Close sets will be returned.

```
Value1 = Mode(Close, 21, 1);
```

Assigns to Value2 the most frequently occurring Range value in the last 30 bars. If there happens to be multiple sets of frequently occurring Range values, lower valued of the Range sets will be returned.

```
Value2 = Mode(Range, 30, -1);
```

ModeArray (Function)

Disclaimer

The ModeArray function returns the most frequently occurring or repetitive value within the array.

Syntax

```
ModeArray(PriceArray, Size, Type)
```

Returns (Double)

A numeric value containing the most frequently occurring or repetitive value within the array. If there is no repetition in the range of data or the Size is greater than the referenced Array, the function will return a -1.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing bar values used to calculate the mode value.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.
Type	Numeric	Sets whether to return the largest or smallest mode value if multiple modes exist (1=highest mode, -1=lowest mode)

Remarks

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The ModeArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

The following assigns the lowest value of the 20 elements in the array to Value1:

```
Array: myArray[20](0);
{… add EL statements here to assign values to array elements... }
Value1 = ModeArray(MyArray, 20);
```

See Also

ExtremesArray, DevSqrdArray

Momentum (Function)

Disclaimer

The Momentum function calculates the change in momentum on the current bar relative to a specified number of bars ago.

Syntax

```
Momentum(Price, Length)
```

Returns (Double)

A numeric value containing Momentum for the current bar.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Length	Numeric	Sets the number of bars to consider.

Remarks

Momentum is an oscillator calculated by subtracting the Price value for the bar occurring Length bars ago from the Price value for the current bar. Therefore, if the value for the current bar exceeds that of the bar in the past, the result will be positive. If the value for the current bar is less than that of the bar in the past, the result will be negative. Values returned by the Momentum function oscillate above and below zero.

Example

Assigns to Value1 the momentum difference of the Close from 10 bars ago.

```
Value1 = Momentum(Close,10);
```

MoneyFlow (Function)

Disclaimer

The MoneyFlow function is an index based on the ratio between positive money flows and all money flows over a specified number of bars.

Syntax

```
MoneyFlow(Length)
```

Returns (Double)

A numeric value containing MoneyFlow for the current bar.

Parameters

Name	Type	Description
Length	Numeric	Sets the number of bars used in calculation.

Remarks

AvgPrice is calculated by taking the average of the Open (if available), Close, High, and Low of a bar. Positive money flow occurs when the current bar's AvgPrice exceeds the previous bar’s AvgPrice, and is calculated by multiplying the current bar’s Volume and its AvgPrice. The positive money flows are then summed over the number of bars specified in Length and divided by the sum of all the money flows specified in Length.

The formula for money flow is:

MoneyFlow = 100 * Sum of Positive MF / Sum of All MF

Example

```
Plot1(MoneyFlow(10), 'M");
```

MRO (Function)

Disclaimer

The MRO (Most Recent Occurrence) function returns the number of bars ago the specified expression was True. Or, if the specified expression did not occur within the last *x* number of bars, the function informs you of such.

Syntax

```
MRO(Test, Length, Instance)
```

Returns (Integer)

A numeric value containing the number of bars ago that the specified `Expression` was `True`; -1 if `Expression` was not found to be `True` within the last `Length` bars.

Parameters

Name	Type	Description
`Test`	TrueFalse	Sets the true/false expression to check for (that is, `Close` > `Open`).
`Length`	Numeric	Sets the number of bars to check.
`Instance`	Numeric	Sets which occurrence, for example, 1 = most recent, 2 = 2nd most recent, and so on.

Remarks

The MRO function is specifically designed to identify when a certain condition occurred. It checks from the current bar and works away from it.

The function returns a number representing how many bars ago the true/false expression was fulfilled (0 = current bar, 1 = one bar ago, 2 = 2 bars ago …). If the function does not find a bar within the specified `Length`, which meets the criteria, it will return a -1.

Note When using the `MRO` function in an analysis technique, always check the result of the `MRO` function before using the value in another calculation or function. If the condition occurred over `Length` number of bars, a -1 value will be returned.

Example

```
Value1 = MRO(Close>Open, 5, 1);
```

See Also

LRO

MyPrice (Function)

Disclaimer

The `MyPrice` function returns the average price of the `High`, `Low` and `Close` prices of the bar.

Syntax

```
MyPrice
```

Returns (Double)

A numeric value containing the average price of the bar.

Parameters

None

Example

Assigns to `Value1` the average bar price of 5 bars ago.

```
Value1 = MyPrice[5];
```

Next3rdFriday (Function)

Disclaimer

The `Next3rdFriday` function calculates the number of calendar days to the next 3rd Friday of a month.

Syntax

```
Next3rdFriday(Series)
```

Returns (Integer)

A numeric value containing the number of calendar days to the 3rd Friday of the Nth month ahead.

Parameters

Name	Type	Description
`Series`	Numeric	Sets the months (ahead) to get the next 3rd Friday. 0=current month, 1=next month, and so on.

Remarks

For the current month, Next3rdFriday(0) will return a negative number of days if the current day of the current month is past the 3rd Friday. In a similar manner, Next3rdFriday(1) returns the number of days to the next 3rd Friday even if it is in the same month as the current day.

Example

If you want to exit out of long positions ten days before the 3rd Friday of the coming month, you can write

```
If Next3rdFriday(1) <= 10 Then
 Sell Next Bar at Market;
```

NormCumDensity (Function)

Disclaimer

The NormCumDensity function calculates the normal density (also called distribution) for the specified mean and standard deviation.

Syntax

```
NormalCumDensity(Price, Length)
```

Returns (Double)

A numeric value containing the normal cumulative density for the specified mean and standard deviation.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to be considered.
Length	Numeric	Sets the number of bars to be considered.

Remarks

This function has a very wide range of applications in statistics, including hypothesis testing.

Example

Assigns to Value1 the Normal Cumulative Density of the current Close based on the last 20 bars.

```
Value1 = NormalCumDensity(Close, 20);
```

NormCumDensityArray (Function)

Disclaimer

The NormCumDensityArray function calculates the normal cumulative density (also called distribution) for an array of values, based on the array-based mean and standard deviation calculations.

Syntax

```
NormalCumDensityArray(PriceArray, Size)
```

Returns (Double)

A numeric value containing the normal cumulative density (also called distribution) for an array of values.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing values upon which the normal cumulative density is calculated.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.

Remarks

This function has a very wide range of applications in statistics, including hypothesis testing.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The NormCumDensityArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Assigns to Value1 the 10 bar normal cumulative density for a specified array.

```
Array: myArray[10](0);
{… (assign values to array) }
Value1 = NormCumDensityArray(myArray, 10);
```

NormDensity (Function)

Disclaimer

The NormDensity function calculates the normal density (also called distribution) for a specific value.

Syntax

```
NormDensity(Price, Length)
```

Returns (Double)

A numeric value containing the normal density (also called distribution) for a specific value.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to be considered.
Length	Numeric	Sets the number of bars to be considered.

Remarks

This function has a very wide range of applications in statistics, including hypothesis testing.

Example

Assigns to Value1 the normal density of the current Close based on the last 20 bars.

```
Value1 = NormDensity(Close, 20);
```

NormDensityArray (Function)

Disclaimer

The NormDensityArray function calculates the normal density (also called distribution) for an array of values, based on the array-based mean and standard deviation calculations.

Syntax

```
NormalDensityArray(PriceArray, Size)
```

Returns (Double)

A numeric value containing the normal density (also called distribution) for an array of values.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing values upon which the normal density is calculated.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.

Remarks

This function has a very wide range of applications in statistics, including hypothesis testing.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The NormDensityArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Assigns to Value1 the 10 bar normal density for a specified array.

```
Array: myArray[10](0);
{… (assign values to array) }
Value1 = NormDensityArray(myArray, 10);
```

NormGradientColor (Function)

Disclaimer

The NormGradientColor function returns a gradient color relative to the current bar value over the specified range of bars.

Syntax

```
NormGradientColor(DataSeriesValue, CrossesZero, ColorNormLength, UpColor, DnColor)
```

Returns (Integer)

A numeric color value relative to a range of color gradient values over a specified number of bars.

Parameters

Name	Type	Description
DataSeriesValue	Numeric	Specifies which bar value (price, function, or formula) to use.
CrossesZero	TrueFalse	Set to True if the DataSeries value can ever be negative, otherwise False.
ColorNormLength	Numeric	Set the number of bars to consider for the high and low values
UpColor	Numeric	Sets the highest color value of the gradient.
DnColor	Numeric	Sets the lowest color value of the gradient.

Remarks

If the current bar value is at or near the highest value over the past ColorNormLength bars, the color returned will be close to UpColor. If the current bar's closing value is at or near its lowest closing value over the last n bars, the color value returned will be close to DnColor.

Example

To calculate the gradient color value within the range of colors from Red to Yellow of MyVolume based on the last 20 bars, you would write:

```
Value1 = NormGradientColor(MyVolume,false,20,Yellow,Red);
```

NormSCDensity (Function)

Disclaimer

The `NormSCDensity` function calculates the standard normal cumulative distribution function.

Syntax

```
NormSCDensity(Price)
```

Returns (Double)

A numeric value containing the standard normal cumulative distribution function.

Parameters

Name	Type	Description
`Price`	Numeric	Specifies which bar value (price, function, or formula) to be considered.

Remarks

The distribution has a mean of zero and a standard deviation of one. Use this function in place of a table of standard normal curve areas.

Example

Assigns to `Value1` the Normal Standard Cumulative Density of the current `Close.`

```
Value1 = NormSCDensity(Close) ;
```

NthExtremes (Function)

Disclaimer

The NthExtremes function will return the highest/lowest extreme, second most extreme, third most extreme (and so on) price over a specified number of bars.

Syntax

```
NthExtremes(Price, Length, N, HiLo, oExtremeVal, oExtremeBar)
```

Returns (Integer)

The oExtremeVal and oExtremeBar output parameters return the 'N'th most extreme Price and the number of bars ago it occurred. The Extremes function itself returns a value of 1 if successful or -1 when N or Length is out of range.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to compare for highest or lowest extremes.
Length	Numeric	The number of trailing bars to consider.
N	Numeric	How extreme (that is, 1 = most, 2 = 2nd most, and so on).
HiLo	Numeric	A switch setting that determines whether the function will return the highest or lowest extreme value. 1=Highest, -1=Lowest.
oExtremeVal	Numeric	Output variable that returns the value of the extreme price.
oExtremeBar	Numeric	Output variable that returns the number of bars ago that the extreme value occurred.

Remarks

Just like the Extremes function, the NthExtremes function has the inputs Price and Length. In addition, it also has the input N. This is the input that specifies if it is the second most extreme, third most extreme (and so on) value desired.

Note The Length input in the NthExtremes function should always be a positive value equal to or less than 100. The value for N should not be greater than the value for the input Length.

See About Functions - Multiple Output Function for more information on using output parameters to return values.

Example

Assigns to Value2 the 3rd highest High of the last 30 bars using the oExtremeVal output parameter, and assigns to Value3 the number of bars ago the 3rd highest High occurred using oExtremeBar output parameter. Value1 is assigned a value of 1:

```
vars: oExtremeVal(0),oExtremeBar(0);
Value1 = Extremes(High, 30, 3, 1, oExtremeVal, oExtremeBar);
Value2 = oExtremeVal;
Value3 = oExtremeBar;
```

NthExtremesArray (Function)

Disclaimer

The NthExtremesArray function will return the highest/lowest extreme, second most extreme, third most extreme (and so on) array element, based on the input N.

Syntax

```
NthExtremesArray(PriceArray, Size, N, HiLo, oExtremeVal, oExtremePosRaw)
```

Returns (Integer)

The oExtremeVal and oExtremeBar output parameters return' N'th most extreme high/low value and the number of bars ago it occurred. The NthExtremesArray function itself returns a value of 1.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing bar values to compare for highest and lowest extremes.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.
N	Numeric	Sets which extreme value to search for (that is, 1 = most, 2 = 2nd most, and so on).
HiLo	Numeric	Sets whether the function will return the highest or lowest extreme value. 1=Highest, -1=Lowest.
oExtremeVal	Numeric	Outputs the highest or lowest extreme value found for the range of bars based on the HiLo setting.
oExtremePosRaw	Numeric	Outputs the number of bars ago the extreme value occurred.

Remarks

Just like the ExtremesArray function, the NthExtremesArray function has the inputs PriceArray and Size. In addition, it also has the input N. This is the input that specifies if it is the second most extreme, third most extreme, and so on value desired.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The NthExtremesArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

See Multiple Output Function for more information on using output parameters to return values.

Example

Assigns to Value2 the highest High in the specified named array myArray with 20 elements using the oExtremeVal output parameter, and assigns to Value3 the number of bars ago the highest High occurred using oExtremeBar output parameter. Value1 is assigned a value of 1:

```
Array: myArray[20](0);
Vars: oExtremeVal(0), oExtremePosRaw(0);
{… add EL statements here to assign High values to array elements... }
Value1 = NthExtremesArray (myArray, 20, 1, oExtremeVal, oExtremePosRaw);
Value2 = oExtremeVal;
Value3 = oExtremePosRaw;
```

See Also

ExtremesArray, NthHighestArray, NthLowestArray

NthHighest (Function)

Disclaimer

The NthHighest function will return the highest, second highest, third highest (and so on) price over a specified number of bars.

Syntax

```
NthHighest(N, Price, Length)
```

Returns (Double)

A numeric value containing the 'N'th highest occurrence of Price over the last Length bars..

Parameters

Name	Type	Description
N	Numeric	The occurrence to return. 1 = lowest, 2 = 2nd lowest, and so on.
Price	Numeric	Specifies which bar value (price, function, or formula) to compare for the lowest occurrence.
Length	Numeric	The number of trailing bars to consider.

Remarks

Just like the Highest function, NthHighest has the inputs Price and Length. It also has the input N that allows you to specify if it is the second, third, and so on highest value desired.

Note The Length input in the NthHighest function can only be replaced with a value equal to or less than 100. The value for N should not be greater than the value for the input Length.

Example

Returns the value of the 2nd highest Close that occurred during the past 30 bars.

```
Value1 = NthHighest(2, Close, 30);
```

NthHighestArray (Function)

Disclaimer

The NthHighestArray function will return the highest, second highest, third highest, and so on array element, based on the input N.

Syntax

```
NthHighestArray(PriceArray, Size, N)
```

Returns (Double)

A numeric value containing the 'N'th highest occurrence of the specified array. If N is greater than the declared array size or the Size input, or it is less than zero, the function will return a -1.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing bar values to compare for highest extremes.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.
N	Numeric	Sets which value to search for (that is, 1 = highest 2 = 2nd highest, and so on).

Remarks

Just like the HighestArray function the NthHighestArray function is based on an array. In addition, it also has the input N. This is the input that specifies whether the first, second, third, and so on highest value desired.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The NthHighestArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Assigns to Value2 the 2nd highest Close in the specified named array myArray with 20 elements.

```
Array: myArray[20](0);
```

{… add EL statements here to assign Close values to array elements... }

```
Value1 = NthHighestArray (myArray, 20, 2);
```

See Also

NthExtremesArray, NthLowestArray

NthHighestBar (Function)

Disclaimer

The NthHighestBar function will return the number of bars ago the highest, second highest, third highest (and so on) price occurred.

Syntax

```
NthHighestBar(N, Price, Length)
```

Returns (Integer)

A numeric value containing the number of bars ago the 'N'highest occurrence of Price occurred.

Parameters

Name	Type	Description
N	Numeric	The occurrence to return. 1 = highest, 2 = 2nd highest, and so on.
Price	Numeric	Specifies which bar value (price, function, or formula) to compare for the highest occurrence.
Length	Numeric	The number of trailing bars to consider.

Remarks

Just like the HighestBar function, NthHighestBar has the inputs Price and Length. It also has the input N that allows you to specify if it is the second, third, and so on highest value desired.

Note The Length input in the NthHighestBar function can only be replaced with a value equal to or less than 100. The value for N should not be greater than the value for the input Length..

Example

Returns the number of bars ago that the 2nd highest Close occurred during the past 25 bars.

```
Value1 = NthHighestBar(2, Close, 25);
```

NthLowest (Function)

Disclaimer

The NthLowest function will return the lowest, second lowest, third lowest (and so on) price over a specified number of bars.

Syntax

```
NthLowest(N, Price, Length)
```

Returns (Double)

A numeric value containing the 'N' th lowest occurrence of Price over the last Length bars..

Parameters

Name	Type	Description
N	Numeric	The occurrence to return. 1 = lowest, 2 = 2nd lowest, and so on.
Price	Numeric	Specifies which bar value (price, function, or formula) to compare for the lowest occurrence.
Length	Numeric	The number of trailing bars to consider.

Remarks

Just like the Lowest function, NthLowest has the inputs Price and Length. It also has the input N that allows you to specify if it is the second, third, and so on lowest value desired.

Note The Length input in the NthLowest function can only be replaced with a value equal to or less than 100. The value for N should not be greater than the value for the input Length..

Example

Returns the value of the 3nd lowest Close that occurred during the past 30 bars.

```
Value1 = NthLowest(3, Close, 30);
```

NthLowestArray (Function)

Disclaimer

The NthLowestArray function will return the lowest, second lowest, third lowest, and so on array element, based on the input N.

Syntax

```
NthLowestArray(PriceArray, Size, N)
```

Returns (Double)

A numeric value containing the 'N'th lowest occurrence of the specified array. If N is greater than the declared array size or the Size input, or it is less than zero, the function will return a -1.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing bar values to compare for the lowest extremes.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.
N	Numeric	Sets which value to search for (that is, 1 = lowest, 2 = 2nd lowest, and so on).

Remarks

Just like the LowestArray function the NthLowestArray function is based on an array. In addition, it also has the input N. This is the input that specifies whether the first, second, third, and so on lowest value desired.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The NthLowestArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Assigns to Value2 the 2nd lowest Close in the specified named array myArray with 20 elements.

```
Array: myArray[20](0);
{… add EL statements here to assign Close values to array elements... }
Value1 = NthLowestArray (myArray, 20, 2);
```

See Also

NthExtremesArray, NthHighestArray

NthLowestBar (Function)

Disclaimer

The NthLowestBar function will return the number of bars ago the lowest, second lowest, third lowest (and so on) price occurred.

Syntax

```
NthLowestBar(N, Price, Length)
```

Returns (Integer)

A numeric value containing the number of bars ago the 'N' th lowest occurrence of Price occurred.

Parameters

Name	Type	Description
N	Numeric	The occurrence to return. 1 = lowest, 2 = 2nd lowest, and so on.
Price	Numeric	Specifies which bar value (price, function, or formula) to compare for the lowest occurrence.
Length	Numeric	The number of trailing bars to consider.

Remarks

Just like the LowestBar function, NthLowestBar has the inputs Price and Length. It also has the input N that allows you to specify if it is the second, third, and so on lowest value desired.

Note The Length input in the NthLowestBar function can only be replaced with a value equal to or less than 100. The value for N should not be greater than the value for the input Length.

Example

Returns the number of bars ago that the 2nd lowest Close occurred during the past 21 bars.

```
Value1 = NthLowestBar(2, Close, 21);
```

NumericRank (Function)

Disclaimer

The NumericRank function returns the rank of a specified price in a sorted price series.

Syntax

```
NumericRank(PriceToRank, Price, Length, SortOrder)
```

Returns (Integer)

The rank (position) of PriceToRank in a price series, as defined by Price and Length, that is sorted in either ascending or descending order. The function returns 0 if PriceToRank is greater than the number of values in the price series.

Parameters

Name	Type	Description
PriceToRank	Numeric	Sets the specific price to be ranked within the Price data series.
Price	Numeric	Specifies which bar value (price, function, or formula) to use for the ranking list.
Length	Numeric	Sets the number of bars to consider.
SortOrder	Numeric	Sets the sort order for the data series being used for ranking: 1 = Descending; -1 = Ascending.

Remarks

As part of the ranking process, the function creates a sorted list of values using the specified Price series over the last Length bars. It then looks for the location (rank) of the value PriceToRank in the list. The exact value of PriceToRank must be found in the price series or the function returns -1.

Example

Assigns to Value1 the rank of the current Close over the most recent 5 Closes, sorted in descending order.

```
Value1 = NumericRank(Close, Close, 5, 1);
```

Assigns to Value1 the rank of the previous bar's Range over the most recent 8 bar Ranges, sorted in ascending order.

```
Value1 = NumericRank(Range[1], Range, 8, -1);
```

See Also

NthExtremes, NthHighest, NthLowest

NumericRankArray (Function)

Disclaimer

The NumericRankArray function returns the rank of a specified price in a sorted data series from an array of values.

Syntax

```
NumericRankArray(PriceToRank, PriceArray, Size, SortOrder)
```

Returns (Integer)

The rank (position) of PriceToRank in a array of values that is sorted in either ascending or descending order. The function returns -1 if PriceToRank is not contained within the specified array, or if the Size is greater than the array that is referenced.

Parameters

Name	Type	Description
PriceToRank	Numeric	Sets the specific price to be ranked within the array .
Price	Numeric	Specifies the array to use for the ranking list.
Length	Numeric	Sets the number of array elements to consider.
SortOrder	Numeric	Sets the sort order for the data series being used for ranking: 1 = Descending; -1 = Ascending.

Remarks

As part of the ranking process, the function creates a sorted list of values using the specified Price series over the last Length bars. It then looks for the location (rank) of the value PriceToRank in the list. The exact value of PriceToRank must be found in the price series or the function returns -1.

The NumericRankArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Assigns to Value1 the rank of a number from a range of values in array myArray with 20 elements.

```
Array: myArray[20](0);
{… add EL statements here to a range of values to array elements... }
Value1 = NumericRankArray (24, myArray, 20, 2);
```

NumUnits (Function)

Disclaimer

The NumUnits function returns the number of shares/contracts for a dollar investment.

Syntax

```
NumUnits(Amnt, MinLot)
```

Returns (Integer)

A number of shares/contracts based on the Amnt and MinLot values specified.

Parameters

Name	Type	Description
Amnt	Numeric	Sets the investment amount, in dollars, per trade
MinLot	Numeric	Sets the minimum lot size desired per transaction

Remarks

NumUnits determines the equivalent number of shares/contracts represented by the dollar amount and current price, given a minimum lot size. This allows for determining how many shares/contracts can be traded with a set amount of dollars.

Example

Returns on a $65 stock, if you wanted to invest $15,500 in 100 share lots, you would be able to trade 200 shares:

```
Value1 = NumUnits(15500, 100);
```

Additional Example

In order to base the number of shares purchased on a dollar/lot amount entered as an Input in a Moving Average Crossover entry, you could use the following syntax:

```
Inputs: Amnt(10000), MinLot(50);

Value1 = NumUnits(Amnt, MinLot);

If Close Crosses Above Average(Close, 18) Then
 Buy Value1 Shares This Bar on Close;
```

OBV (Function)

Disclaimer

The OBV function calculates "On Balance Volume".

Syntax

```
OBV
```

Returns (Double)

A numeric value containing OBV for the current bar.

Parameters

None

Remarks

Mathematically, the cumulative OBV formula is represented as:

$OBV = \sum[(C-Cp / |C-Cp|)*V]$

. . . where

C is the current period’s closing price.

Cp is the previous period’s closing price.

|C - Cp| is the absolute value of the difference between the two closing prices.

Since any number divided by itself is equal to one (1), the only bearing the expression in parenthesis has is on the sign. If the previous Close is greater than the current Close, the sign will be negative. If the previous Close is less than the current Close, the sign will be positive.

Therefore, the function either subtracts the current volume from the running total of the OBV line or adds current volume to the running total of the OBV line.

Example

```
Value1 = OBV;
```

Reference

Joseph E. Granville, *A New y of Daily Stock Market Timing for Maximum Profit*, Prentice-Hall, Englewood Cliffs, NJ.

OHLCPeriodsAgo (Function)

Disclaimer

The OHLCPeriodsAgo series function returns the Open, High, Low, and Close of a specified time segment, a specified number of periods ago.

Syntax

```
OHLCPeriodsAgo(PeriodType, PeriodsAgo, oPeriodOpen, oPeriodHigh, oPeriodLow,
oPeriodClose)
```

Returns (Integer)

The oPeriodOpen, oPeriodHigh, oPeriodLow, and oPeriodCLose output parameters returns the Open, High, Low, and Close from a time segment. The OHLCPeriodsAgo function itself returns 1 if the specified time segment was available, and -1 if not.

Parameters

Name	Type	Description
PeriodType	Numeric	Sets the time period on which to base values (1=Day, 2=Week, 3=Month, 4=Year)
PeriodsAgo	Numeric	Sets the number of PeriodType periods ago
oPeriodOpen	Numeric	Outputs the open of the specified time segment
oPeriodHigh	Numeric	Outputs the high of the specified time segment
oPeriodLow	Numeric	Outputs the low of the specified time segment
oPeriodClose	Numeric	Outputs the close of the specified time segment

Remarks

See Multiple Output Function for more information on using output parameters to return values.

Example

Assigns to Value2 through Value5 the Open, High, Low, and Close of the last 20 weeks. Value1 is assigned a value of 1.

```
vars: oPeriodOpen(0), oPeriodHigh(0), oPeriodLow(0), oPeriodClose(0);
Value1 = OHLCPeriodsAgo(2, 20, oPeriodOpen, oPeriodHigh, oPeriodLow, oPeriodClose);
Value2 = oPeriodOpen;
Value3 = oPeriodHigh;
Value4 = oPeriodLow;
Value5 = oPeriodClose;
```

Returns (Integer)

The oExtremeVal and oExtremeBar output parameters return the extreme value and the number of bars ago it occurred. The Extremes function itself returns a value of 1.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to compare for highest and lowest extremes.
Length	Numeric	Sets the number of bars to consider for extremes.
HiLo	Numeric	Sets whether the function will return the highest or lowest extreme value. 1=Highest, -1=Lowest.
oExtremeVal	Numeric	Outputs the highest or lowest extreme value found for the range of bars based on the HiLo setting.

oExtremeBar	Numeric	Outputs the number of bars ago the extreme value occurred.

Remarks

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as: (High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

See Multiple Output Function for more information on using output parameters to return values.

Example

Assigns to Value2 the highest High of the last 20 bars using the oExtremeVal output parameter, and assigns to Value3 the number of bars ago the highest High occurred using oExtremeBar output parameter. Value1 is assigned a value of 1:

```
vars: oExtremeVal(0), oExtremeBar(0);
Value1 = Extremes(High, 20, 1, oExtremeVal, oExtremeBar);
Value2 = oExtremeVal;
Value3 = oExtremeBar;
```

OneAlert (Function)

Disclaimer

The `OneAlert` function is used to limit alerts to one per bar.

Syntax

```
OneAlert(Criteria)
```

Returns (Boolean)

Returns true only on the first tick on a bar that also matches the specified *Criteria*.

Parameters

Name	Type	Description
Criteria	TrueFalse	True-False expression that must match for the function to return a "true" value.

Remarks

The function returns true only on the first tick in each bar on which the function is called and on which the statement contained in the input "Criteria" evaluates to true. Returns False when called on subsequent ticks in that bar, regardless of whether "Criteria" evaluates to true or to false on the subsequent ticks. Repeats process with each new bar.

Example

```
Condition1 = OneAlert(Close > Close[1]);
```

OpenD (Function)

Disclaimer

The OpenD series function allows you to reference the daily open of a previous day in an intraday chart (minute or tick-based) or a daily chart. OpenD is one of a family of functions that allows historical references across various data intervals.

Syntax

```
OpenD(PeriodsAgo)
```

Returns (Double)

The daily open price from a specified number of days ago. If the PeriodsAgo parameter is out of range (> 50), or there is not enough data, the function will return –1.

Parameters

Name	Type	Description
PeriodsAgo	Numeric	Sets the number of days/periods back to reference a previous day's open price. (50 days back maximum) (0 = Today's current high)

Remarks

You must have enough intraday data in the chart in order to look back and reference any previous daily open. For example, if you want to look back at the high of 25 days ago on a 5-minute chart, you must have at least 26 full days of 5-minute bars in the chart.

The value for the PeriodsAgo input parameter should always be a whole number greater than or equal to 0, but less than 51. Setting PeriodsAgo to 0 returns today's current open.

Example

In order to place a buy limit order at the open of the previous day you would write:

```
Buy Next Bar at OpenD(1) Limit;
```

See Also

LowD, CloseD, HighD, HighW, HighM, and HighY.

OpenM (Function)

Disclaimer

The OpenM series function allows you to reference the monthly Open of a previous month in an intraday chart (minute or tick-based) or a daily, weekly, or monthly chart. OpenM is one of a family of functions that allows historical references across various data intervals.

Syntax

```
OpenM(PeriodsAgo)
```

Returns (Double)

The monthly Open price from a specified number of months ago. If the PeriodsAgo parameter is out of range (> 50), or there is not enough data, the function will return –1.

Parameters

Name	Type	Description
PeriodsAgo	Numeric	Sets the number of months back to reference a previous month's Open price. (50 months back maximum) (0 = This month's current Open)

Remarks

You must have enough intraday data in the chart in order to look back and reference any previous monthly Open. For example, if you want to look back at the Open of 11 months ago on a 60-minute chart, you must have at least 12 full months of 60-minute bars in the chart.

The value for the PeriodsAgo input parameter should always be a whole number greater than or equal to 0, but less than 51. Setting PeriodsAgo to 0 returns this month's current Open.

Example

In order to place a buy limit order at the Open of the previous month you would write:

```
Buy Next Bar at OpenM(1) Limit;
```

See Also

LowM, CloseM, HighM, HighD, HighW and HighY.

OpenW (Function)

Disclaimer

The OpenW series function allows you to reference the weekly Open of a previous week in an intraday chart (minute or tick-based) or a daily or weekly chart. OpenW is one of a family of functions that allows historical references across various data intervals.

Syntax

```
OpenW(PeriodsAgo)
```

Returns (Double)

The weekly open price from a specified number of weeks ago. If the PeriodsAgo parameter is out of range (> 50), or there is not enough data, the function will return –1.

Parameters

Name	Type	Description
PeriodsAgo	Numeric	Sets the number of weeks back to reference a previous week's open price. (50 weeks back maximum) (0 = This week's current open)

Remarks

You must have enough intraday data in the chart in order to look back and reference any previous weekly open. For example, if you want to look back at the open of 15 weeks ago on a 30-minute chart, you must have at least 16 full weeks of 30-minute bars in the chart.

The value for the PeriodsAgo input parameter should always be a whole number greater than or equal to 0, but less than 51. Setting PeriodsAgo to 0 returns this week's current open.

Example

In order to place a buy limit order at the open of the previous week you would write:

```
Buy Next Bar at OpenW(1) Limit;
```

See Also

LowW, CloseW, HighW, HighD, HighM and HighY.

OpenY (Function)

Disclaimer

The OpenY series function allows you to reference the yearly open of a previous year in an intraday chart (minute or tick-based) or a daily or yearly chart. OpenY is one of a family of functions that allows historical references across various data intervals.

Syntax

```
OpenY(PeriodsAgo)
```

Returns (Double)

The yearly open price from a specified number of years ago. If the PeriodsAgo parameter is out of range (> 50), or there is not enough data, the function will return –1.

Parameters

Name	Type	Description
PeriodsAgo	Numeric	Sets the number of years back to reference a previous year's open price. (50 years back maximum) (0 = This year's current open)

Remarks

You must have enough intraday data in the chart in order to look back and reference any previous yearly open. For example, if you want to look back at the open of 7 years ago on a daily chart, you must have at least 8 full years of daily bars in the chart.

The value for the PeriodsAgo input parameter should always be a whole number greater than or equal to 0, but less than 51. Setting PeriodsAgo to 0 returns this week's current open.

Example

In order to place a buy limit order at the open of the previous year you would write:

```
Buy Next Bar at OpenY(1) Limit;
```

See Also

LowY, CloseY, HighY, HighD, HighW and HighM.

OptionComplex (Function)

Disclaimer

The `OptionComplex` function calculates the theoretical value of an option and its Greek risk values based on the Black-Scholes option pricing model.

Syntax

```
OptionComplex(MyAssetType, DaysLeft, StrikePr, AssetPr, Rate100, Yield100,
ForeignRate100, Volty100, PutCall, EuroAmer01, oOptPrice, oDelta, oGamma, oVega,
oTheta, oRho)
```

Returns (Double)

The oOptPrice, oDelta, oGamma, oVega, oTheta, and oRho output parameters return the theoretical option price and its related Greek risk values. The OptionComplex function itself returns a value of 1.

Parameters

Name	Type	Description
`MyAssetType`	Numeric	Sets the underlying asset type to: 1 = Non Dividend-Paying Stock 2 = Dividend-Paying Stock 3 = Futures 4 = Currencies
`DaysLeft`	Numeric	Sets the number of calendar days left for the option.
`StrikePr`	Numeric	Specifies the strike price of the option.
`AssetPr`	Numeric	Specifies the price of the underlying asset.
`Rate100`	Numeric	Sets the short-term risk free interest rate, usually the 90-day T-Bill, as a percentage (enter 4.9% as 4.9).
`Yield100`	Numeric	Sets the dividend yield rate as a percentage (enter 1% as 1).
`ForeignRate100`	Numeric	Sets the foreign risk free interest rate as a percentage (enter 3% as 3).
`Volty100`	Numeric	Sets the volatility of the underlying asset as a percentage (enter 22.5% as 22.5).
`PutCall`	Numeric	Specifies if it is a Put or Call option. Put or 2 = Puts; Call or 3 = Calls.
`EuroAmer01`	Numeric	Specifies if the option is a European or American option. 0 = European; 1 = American.
`oOptPrice`	Numeric	Outputs the option theoretical price.
`oDelta`	Numeric	Outputs Delta, which measures the option's price sensitivity to a change in the price of the underlying asset.
`oGamma`	Numeric	Outputs Gamma, which measures the acceleration of Delta for every one dollar price movement of the underlying security.
`oVega`	Numeric	Outputs Vega, which measures the option's price sensitivity to the volatility of the underlying security.
`oTheta`	Numeric	Outputs Theta, which measures the option's price sensitivity to its expiration date.
`oRho`	Numeric	Outputs Rho, which measures the option's price sensitivity to the applicable interest rate.

Remarks

The input parameter DaysLeft can also be a numeric function such as DaystoExpiration or Next3rdFriday.
The input parameter Volty100 can also use a reserved word value such as IVolatility *100.
The input parameter ForeignRate100 will only be considered when pricing for a currency option.

See Multiple Output Function for more information on using output parameters to return values.

Example

Assigns to Value2 the theoretical price of a European Call option using the oOptPrice output parameter. Value1 is assigned a value of 1:

```
Vars: oOptPrice(0),oDelta(0),oGamma(0),oVega(0),oTheta(0),oRho(0);

Value1 = OptionPrice(1, DaystoExpiration(1, 107), Strike, Close, 4.9, 0, 0, 22.5,
Call, 0, oOptPrice, oDelta, oGamma, oVega, oTheta, oRho);

Value2 = oOptPrice;
```

OptionPrice (Function)

Disclaimer

The OptionPrice function calculates the theoretical value of an option based on the Black-Scholes option pricing model. OptionPrice provides added flexibility over the BlackScholes and Black functions by allowing you to specify the underlying asset type, the dividend yield for a dividend-paying stock, the foreign risk free interest rate, and if the calculation is for a European or American option.

Syntax

```
OptionPrice(MyAssetType, DaysLeft, StrikePr, AssetPr, Rate100, Yield100,
ForeignRate100, Volty100, PutCall, EuroAmer01)
```

Returns (Double)

A numeric value representing the theoretical value of the specified option.

Parameters

Name	Type	Description
MyAssetType	Numeric	Sets the underlying asset type to: 1 = Non Dividend-Paying Stock 2 = Dividend-Paying Stock 3 = Futures 4 = Currencies
DaysLeft	Numeric	Sets the number of calendar days left for the option.
StrikePr	Numeric	Specifies the strike price of the option.
AssetPr	Numeric	Specifies the price of the underlying asset.
Rate100	Numeric	Sets the short-term risk free interest rate, usually the 90-day T-Bill, as a percentage (enter 4.9% as 4.9).
Yield100	Numeric	Sets the dividend yield rate as a percentage (enter 1% as 1).
ForeignRate100	Numeric	Sets the foreign risk free interest rate as a percentage (enter 3% as 3).
Volty100	Numeric	Sets the volatility of the underlying asset as a percentage (enter 22.5% as 22.5).
PutCall	Numeric	Specifies if it is a Put or Call option. Put or 2 = Puts; Call or 3 = Calls.
EuroAmer01	Numeric	Specifies if the option is a European or American option. 0 = European; 1 = American.

Remarks

The input parameter DaysLeft can also be a numeric function such as DaystoExpiration or Next3rdFriday.
The input parameter Volty100 can also use a reserved word value such as IVolatility *100.
The input parameter ForeignRate100 will only be considered when pricing for a currency option.

Example

Assigns to Value1 the theoretical price of a European Call option using the DaystoExpiration function to calculate the number of days left in the option until it expires in January of 2007 from today with the short-term 90-day T-Bill at 4.9%.

```
Value1 = OptionPrice(1, DaystoExpiration(1, 107), Strike, Close, 4.9, 0, 0, 22.5,
Call, 0);
```

See Also

BlackModel, BlackSholes.

OS_AnnualDividend (Function)

Disclaimer

The OS_AnnualDividend function calculates the dividend adjustment to the asset price. This function is designed primarily for OptionStation.

Syntax

```
OS_AnnualDividend(AnnualDividend, ExpirDays, InterestRate) ;
```

Returns (Double)

A numeric value representing the dividend adjustment to the asset price.

Parameters

Name	Type	Description
AnnualDividend	Numeric	Sets Amount of dividend, expressed in dollars (enter 1.25 to represent $1.25)
ExpirDays	Numeric	Sets the number of calendar days left until expiration of option.
InterestRate	Numeric	Sets the short-term risk free interest rate as a percentage, usually for 90-day T-Bill (enter 4.9 for 4.9%)

Remarks

The OptionStation BS (Black Scholes) Annual Dividend Pricing Model uses the OS_AnnualDividend function to calculate the dividend adjustment in order to calculate the theoretical option price and Greek values.

Example

Assigns Value1 the annual dividend adjustment for a stock that will pay a dividend of 1.25 in 30 days when the risk free interest rate is 5.0%.

```
Value1 = OS_AnnualDividend(1.25, 30, 5.0);
```

OS_Binomial (Function)

Disclaimer

The OS_Binomial function is used to calculate the theoretical value of on option using the Binomial option pricing algorithm. This function is designed for OptionStation.

Syntax

```
OS_Binomial(AssetPr, StrikePr, Volty, DaysToExp, IntRate, Carry, OptType,
EuroAmer01, Branches, oBinTV) ;
```

Returns (Double)

A numeric value of 0 if successful and -1 if not successful. The oBinoTV output parameter returns the theoretical value of the option.

Parameters

Name	Type	Description
AssetPr	Numeric	Specifies the price of underlying asset.
StrikePr	Numeric	Specifies the strike price of the option.
Volty	Numeric	Volatility of the underlying asset expressed as a percentage.
DaysToExp	Numeric	Days left until expiration of option.
IntRate	Numeric	Risk free short-term interest rate, usually the 90-day T-Bill rate.
Carry	Numeric	Broker fees for margin accounts.
OptType	Numeric	Specifies type of option: 2 for a Put, 3 for a Call.
EuroAmer01	Numeric	Type of options where 0 = European Options and 1 = American options.
Branches	Numeric	Number of binomial branches (iterations) the function calculates. The more branches used, the slower the calculation. For end of day values, use 50; for real time values, use 10 to 20.
oBinoTV	Numeric	Outputs the theoretical value of the option.

Example

Assigns Value1 the theoretical price of an option with an underlying asset price of 97.25, strike price of 10, volatility of 32.5%, 30 days till expiration, risk free interest rate of 5.0%, a carrying cost of 5.0%, is a Call, calculation for an American option, using 50 calculation branches (iterations).

```
Vars: oBinTV(0);
Value1 = OS_Binomial(97.250,100,32.5,30,5.0,5.0,Call,”1”,50, oBinTV);
```

OS_CheckProx (Function)

Disclaimer

The OS_CheckProx function checks to see if the target strike price is within the proximity specified. This function is designed for OptionStation.

Syntax

```
   OS_CheckProx(OptType, OptExpDate, InOrOut01, Prox, CheckStrike);
```

Returns (Boolean)

True if the target strike price is within the proximity specified, otherwise False if not found.

Parameters

Name	Type	Description
OptType	Numeric	Option type is a numerical value of either 0 or 2 for a Put, or 1 or 3 for a Call.
OptExpDate	Numeric	Sets the target expiration date.
InOrOut01	Numeric	Sets whether the function should check if option is in-the-money or out-of-the money (0 = In-the-Money, 1 = Out-of-the-Money)
Prox	Numeric	Sets the number of strikes to use for proximity test.
CheckStrike	Numeric	Sets the strike price to test.

Remarks

This function is used in many OptionStation search strategies to allow the user to filter out strikes too far in or out of the money in position search.

Example

Assigns Condition1 to true if the strike price is within the proximity for a 100 Call option, that is 3 strikes or less in the money.

```
   Condition1 = OS_CheckProx(Call, ExpirationDate of Option, 1, 3, 100);
```

OS_DaysToExp (Function)

Disclaimer

The OS_DaysToExp function calculates the days to expiration in whole or fractional days. This function is designed primarily for use in OptionStation.

Syntax

```
OS_DaysToExp(TargetExpDate, TargetCalcDate, TargetCalcTime, UseFractionalDays);
```

Returns (Double)

A numeric value representing the days to expiration in whole or fractional days for a position.

Parameters

Name	Type	Description
TargetExpDate	Numeric	Sets the expiration date of the option in Julian Date format.
TargetCalcDate	Numeric	Sets the modeled or today's date in Julian date format.
TargetCalcTime	Numeric	Sets the current time in minutes from midnight.
UseFractionalDays	Numeric	Sets whether the function will return the days to expiration in whole or fractional days. True = results to be displayed in fractional days; False = results to be in whole days.

Remarks

The input parameters TargetExpDate can also use the reserved word value such as ExpirationDate.
The input parameters TargetCalcDate can also use the reserved word value such as LastCalcJDate.

Example

Assigns to Value1 the days to expiration of an option, in fractional portions, from today, at the current time

```
Value1 = OS_DaysToExp(ExpirationDate of Option, LastCalcJDate, LastCalcMMTime,
True);
```

OS_DaysToFarExp (Function)

Disclaimer

The OS_DaysToFarExp function calculates the days to the far expiration for a position. This function is designed primarily for use within the OptionStation Analysis window.

Syntax

```
OS_DaysToFarExp
```

Returns (Double)

A numeric value that is the days to the far expiration in fractional days.

Parameters

None

Remarks

This function is useful when you're trying to find the far expiration of a position that my contain different strategies or legs with different expirations.

Example

Plots the days to the far expiration for a position, for the current option, from today, at the current time, and calculate the fractional portion.

```
Plot1(OS_DaysToFarExp);
```

OS_DaysToNearExp (Function)

Disclaimer

The OS_DaysToNearExp function calculates the days to the near expiration for a position. This function is designed primarily for use within the OptionStation Analysis window.

Syntax

```
OS_DaysToNearExp
```

Returns (Double)

A numeric value that is the days to the near expiration in fractional days.

Parameters

None

Remarks

This function is useful when you're trying to find the near expiration of a position that my contain different strategies or legs with different expirations.

Example

Plots the days to the near expiration for a position, for the current option, from today, at the current time, and calculate the fractional portion.

```
Plot1(OS_DaysToNearExp);
```

OS_DivsBetweenDates (Function)

Disclaimer

The OS_DivsBetweenDates function calculates the present value of the dividends that are to be paid between two user-specified dates. This function is designed primarily for use in OptionStation.

Syntax

```
OS_DivsBetweenDates(CurrDate, FutureDate, FirstDivMonth, FirstDivDay, DivAmt, Rate,
PV);
```

Returns (Double)

A numeric value representing the present value of the dividends that are to be paid between two user-specified dates, the date input as CurrDate and the date input as FutureDate.

Parameters

Name	Type	Description
CurrDate	Numeric	Sets today's date using EasyLanguage date format, YYYMMDD (enter 1061212 for Dec 12 2006).
FutureDate	Numeric	Sets the expiration date of an option using EasyLanguage date format, YYYMMDD (enter 1061212 for Dec 12 2006).
FirstDivMonth	Numeric	Sets the month in which the first annual dividend payment is due. Enter 1 for January, 2 for February and so on.
FirstDivDay	Numeric	Sets the day of the month on which the first annual dividend payment is due (enter 15 if the due date is January 15th).
DivAmt	Numeric	Sets the amount of dividend (enter 0.0125 for 1.25%).
Rate	Numeric	Sets the short-term risk free interest rate, usually the 90-day T-Bill (enter .049 for 4.9%).
PV	TrueFalse	Sets whether the function will calculate the dividend in present value. True = Calculate present value adjustment, and False = Do not calculate present value adjustment.

Example

Assigns to Value1 the periodic dividend adjustment for a stock that will pay a dividend of 1.25 on December 15, and do not calculate the present value adjustment.

```
Value1 = OS_DivsBetweenDates(1060803, 1061117, 12, 15, 1.25, False);
```

OS_FindCall (Function)

Disclaimer

The OS_FindCall function returns the ID number for the specified Call option. This function is designed primarily for use in OptionStation.

Syntax

```
OS_FindCall(SeriesProx, StrikeProx)
```

Returns (Double)

A numeric value representing the ID number for the specified Call option.

Parameters

Name	Type	Description
SeriesProx	Numeric	Sets the option series proximity to the near series, where 1 = near series, 2 = next trading month and so on.
StrikeProx	Numeric	Sets the option strike price proximity to the At-The-Money strike price. 0 = At-The-Money strike price 1 = next Out-Of-The-Money strike price -1 = next In-The-Money strike price

Remarks

The ID number of an option in an option(n) array for a Call is based on a specific series and strike price proximity. OptionStation maintains an array of all options in the analysis, which you can referenced by using the “of option(n)” convention.

Example

Assigns to Value1 the strike price of a near series, At-The-Money Call option.

```
Value1 = Strike of option(OS_FindCall(1,0)) ;
```

OS_FindPut (Function)

Disclaimer

The OS_FindPut function returns the ID number for the specified Put option. This function is designed primarily for use in OptionStation.

Syntax

```
OS_FindPut(SeriesProx, StrikeProx)
```

Returns (Double)

A numeric value representing the ID number for the specified Put option.

Parameters

Name	Type	Description
SeriesProx	Numeric	Sets the option series proximity to the near series, where 1 = near series, 2 = next trading month and so on.
StrikeProx	Numeric	Sets the option strike price proximity to the At-The-Money strike price. 0 = At-The-Money strike price 1 = next Out-Of-The-Money strike price -1 = next In-The-Money strike price

Remarks

The ID number of an option in an option(n) array for a Put is based on a specific series and strike price proximity. OptionStation maintains an array of all options in the analysis, which you can referenced by using the “of option(n)” convention.

Example

Assigns to Value1 the strike price of a near series, At-The-Money Put option.

```
Value1 = Strike of option(OS_FindPut(1,0)) ;
```

OS_FindSeries (Function)

Disclaimer

The `OS_FindSeries` function returns the option series proximity to the near series. This function is designed primarily for use in OptionStation.

Syntax

```
OS_FindSeries(OptType, ExpDate)
```

Returns (Double)

A numeric value representing the option series proximity to the near series.

Parameters

Name	Type	Description
`OptType`	Numeric	Sets whether it is a Put or Call option. Put or 2 = *Puts*; Call or 3 = *Calls*.
`ExpDate`	Numeric	Sets the expiration date of the option in EasyLanguage format (enter December 1 2006 as 1061201).

Remarks

This function can be used to set the series proximity input in the OS_FindCall and OS_FindPut functions.

Example

Assigns to Value1 the series proximity value to the near series month for the option.

```
Value1 = OS_FindSeries(Call, 1070199) ;
```

OS_FracDaysToExp (Function)

Disclaimer

The OS_FracDaysToExp function returns the fractional portion remaining in a current session. This function is designed primarily for use in OptionStation.

Syntax

```
OS_FracDaysToExp(TargetCalcTime)
```

Returns (Double)

A numeric value representing the fractional portion of the session at the point of the target time specified.

Parameters

Name	Type	Description
TargetCalcTime	Numeric	Sets the current time of the bar in minutes from midnight.

Remarks

The fractional portion of the session at the point of the target time is subtracted from the whole days to expiration, to more accurately calculate intra-day value for days to expiration.

Example

Assigns to Value1 the fractional portion remaining in the current session. You can subtract the fractional value from the whole days to get a more accurate intra-day value for days to expiration.

```
Value1 = OS_ FracDaysToExp(LastCalcMMTime);
Value2 = DaysToExpiration - Value1;
```

OS_GrossIn (Function)

Disclaimer

The `OS_GrossIn` function calculates the gross entry price for an option position created in the Position section the OptionStation Analysis window. This function is designed primarily for use within the OptionStation Analysis window.

Syntax

```
OS_GrossIn
```

Returns (Double)

A numeric value containing the gross entry price (stated as a negative value) for an option position created in the Position section the OptionStation Analysis window.

Parameters

None

Example

```
Plot1(OS_GrossIn, “EntryPrice”);
```

OS_GrossOut (Function)

Disclaimer

The `OS_GrossOut` function calculates the gross exit price for an option position created in the Position section the OptionStation Analysis window. This function is designed primarily for use within the OptionStation Analysis window.

Syntax

```
OS_GrossOut
```

Returns (Double)

A numeric value containing the gross exit price for an option position created in the Position section the OptionStation Analysis window.

Parameters

None

Example

```
Plot1(OS_GrossOut, “ExitPrice”);
```

OS_Intrinsic (Function)

Disclaimer

The OS_Instrinsic function calculates the difference between the option's strike price and the price of the underlying asset, which determines how much an option is in-the-money. This function is designed primarily for use in OptionStation.

Syntax

```
OS_Intrinsic(AssetPr,OptType,StrikePr)
```

Returns (Double)

A numeric value representing the intrinsic value of an option..

Parameters

Name	Type	Description
AssetPr	Numeric	Specifies the price of the underlying asset
OptType	Numeric	Sets if it is a Put or Call option. Put or 2 = Puts; Call or 3 = Calls.
StrikePr	Numeric	Sets the strike price of option.

Remarks

Intrinsic value represents the current ITM (In The Money) value of the option. Intrinsic value is also the difference between the option premium and time value.

Example

Assigns to Value1 the intrinsic value of a Call option with a strike price of 55 and an underlying asset price of 62.125:

```
Value1 = OS_Intrinsic(62.125,1,55);
```

Assigns to Value2 the intrinsic value of a Put option with a strike price of 105 and an underlying asset price of 98.125:

```
Value2 = OS_Intrinsic(98.125,0,105);
```

OS_MaxNumStrikes (Function)

Disclaimer

The OS_MaxNumStrikes function returns the maximum number of in- and out-of-the-money options based on the given series and option type. This function is designed primarily for use with OptionStation.

Syntax

```
OS_MaxNumStrikes(OptType, TargetExpDate, oIn, oOut);
```

Returns (Double)

The oIn and oOut output parameters return the maximum number of in-the-money and out-of-the money options for a given series and option type. The OS_MaxNumStrikes function itself returns a value of 0 if successful, and -1 otherwise.

Parameters

Name	Type	Description
OptType	Numeric	Sets if it is a Put or Call option. Put or 2 = Puts; Call or 3 = Calls.
TargetExpDate	Numeric	Sets the expiration date of the option in Julian Date format.
oIn	Numeric	Outputs the maximum number of in-the-money options based on the given series and option type.
oOut	Numeric	Outputs the maximum number of out-of-the-money options based on the given series and option type

Example

Sets the variables Value21 and Value3 to the number of strikes In and Out of the money for a given option type and series.

```
Value1 = OS_MaxNumStrikes(Call, ExpirationDate of Option, oIn, oOut);
Value2 = oIn;
Value3 = oOut;
```

Parabolic (Function)

Disclaimer

The Parabolic series function returns the parabolic stop/reversal for the current bar.

Syntax

```
Parabolic(AfStep)
```

Returns (Double)

A numeric value containing the parabolic stop/reversal for the current bar.

Parameters

Name	Type	Description
AfStep	Numeric	Sets the acceleration factor increment, generally set to 0.02.

Remarks

This function is based on a relationship between time and price. Based on the first bar of a chart, the function first calculates a value at an extreme reference point above a bar, and as each bar progresses, the value calculated will approach each bar progressively. Once a calculated value falls within the High-Low range of a bar, the next value will be calculated at an extreme reference point below the next bar. Again, the incremental approach will resume from the bottom up, alternating again when the calculated value falls within the High-Low range of a bar.

The expression ‘parabolic’ derives from the shape of the curve the values create as the results of this function are plotted on a chart.

This function provides the calculations necessary to generate order(s) for the Parabolic trading strategy described in Welles Wilder’s book *New Concepts In Technical Trading Systems*.

The value returned by this function is the Parabolic stop value on the current bar. See ParabolicSAR for more information.

Example

Plots the parabolic close value based on a .02 increment::

```
Plot1(Parabolic(.02));
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC

ParabolicCustom (Function)

Disclaimer

The ParabolicLimit series function returns the parabolic stop/reversal for the current bar.

The ParabolicCustom function provides greater flexibility than the Parabolic by allowing you to set the acceleration limit factor. The greater the AfLimit value, the longer the Parabolic calculation will continue to accelerate.

Syntax

```
ParabolicCustom(AfStep, AfLimit)
```

Parameters

Name	Type	Description
AfStep	Numeric	Sets the acceleration factor increment, generally set to 0.02.
AfLimit	Numeric	Sets the acceleration limitation factor.

Returns (Double)

A numeric value representing parabolic stop value for the current bar.

Remarks

This function is based on a relationship between time and price. Based on the first bar of a chart, the function first calculates a value at an extreme reference point above a bar, and as each bar progresses, the value calculated will approach each bar progressively. Once a calculated value falls within the High-Low range of a bar, the next value will be calculated at an extreme reference point below the next bar. Again, the incremental approach will resume from the bottom up, alternating again when the calculated value falls within the High-Low range of a bar.

The expression ‘parabolic’ derives from the shape of the curve the values create as the results of this function are plotted on a chart.

The value returned by this function is the Parabolic value on the current bar. See ParabolicSAR for more information.

Example

Plots the parabolic close value based on a .025 increment::

```
Plot1(ParabolicCustom(.025));
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC

ParabolicSAR (Function)

Disclaimer

The ParabolicSAR series function returns the parabolic stop values for the current bar and the next bar, as well as the probable position of the market.

Syntax

```
ParabolicSAR(AfStep, AfLimit, oParCl, oParOp, oPosition, oTransition)
```

Parameters

Name	Type	Description
AfStep	Numeric	Sets the acceleration increment factor.
AfLimit	Numeric	Sets the acceleration limitation factor.
oParCl	Numeric	Outputs the parabolic stop value for the current bar.
oParOp	Numeric	Outputs the parabolic stop value for the next bar.
oPosition	Numeric	Outputs the potential market position. 1 for long, -1 for short.
oTransition	Numeric	Outputs if the current bar market position has changed. 1 or -1 for reversal days 0 for holding days.

Returns (Integer)

The oParCl, OParOP, oPosition, oTransition output parameters return the parabolic stop values for the current day and the next bar, as well as the probable position of the market. The ParabolicSAR itself returns 1.

Remarks

This function is based on a relationship between time and price. Based on the first bar of a chart, the function first calculates a value at an extreme reference point above a bar, and as each bar progresses, the value calculated will approach each bar progressively. Once a calculated value falls within the High-Low range of a bar, the next value will be calculated at an extreme reference point below the next bar. Again, the incremental approach will resume from the bottom up, alternating again when the calculated value falls within the High-Low range of a bar.

The expression ‘parabolic’ derives from the shape of the curve the values create as the results of this function are plotted on a chart.

The input oParCl returns the calculation of Parabolic for the current bar and is useful when plotting a study. oParOp returns the calculation for the next bar and is most useful when incorporated into a strategy. For example, this function provides the calculations necessary to generate order(s) for the Parabolic trading strategy described in Welles Wilder’s book *New Concepts In Technical Trading Systems*.

Example

```
Vars: oParCl(0), oParOp(0), oPostion(0), oTransition(0);

Value1 = ParabolicSAR(0.02, 0.4, oParCl, oParOp, oPosition, oTransition);
Value2 = oParCl;
Value3 = oParOp;
Value4 = oPosition;
Value5 = oTransition;
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC

PartSessionCountDay (Function)

Disclaimer

The SPartSessionCountDay function returns the number of partial sessions for the specified day. A partial session is defined as a session that does not start and end on the same day. There can be a maximum of two partial sessions for any given day.

Syntax

```
SessionCountDay(SessionType,XDay);
```

Returns (Integer)

A numeric value for the current bar.

Parameters

Name	Type	Description
SessionType	Numeric	Sets the type of session to reference. 0 = Auto Detect, 1 = Regular Session
XDay	Numeric	Sets the day of the week that should be evaluated. 0=Sunday, 1=Monday, etc.

Remarks

The input parameter SessionType specifies the type of session information that should be returned. The type parameter may be specified as follows: Auto-Detect Session – [0, AutoSession] - uses whatever session is specified for the data series; regular or custom, or Regular Session – [1, RegularSession] - regular session information should always be returned.

Examples

Assigns to Value1 the number of partial sessions for Wednesday:

```
Value1 = SessionCountDay(0,3);
```

See Also

SessionCountDay

Pennant (Function)

Disclaimer

The Pennant series function returns the start and end prices of the current pennant High and Low lines.

Syntax

```
Value1 = Pennant(Length, MaxConsolIndex, BarsPast, oTLHiStartPr, oTLHiEndPr,
oTLLoStartPr, oTLLoEndPr);
```

Returns

The `oTLHiStartPr, oTLHiEndPr, TLLoStartPr,` and `oTLLoEndPr` output parameters return the start and end prices of the current pennant high and low.

The `Pennant` function itself returns:

- 1 if a pennant has just been identified. (The start and end prices of the defining lines are returned as outputs.)
- 2 if the price has broken out above the most recently completed pennant within the previous BarPast bars.
- 3 if the price has broken out below the most recently completed pennant within the previous BarPast bars.
- -1 if none of the above.

Parameters

Name	Type	Description
`Length`	Numeric	Sets the number of bars in the pennant.
`MaxConsolIndex`	Numeric	See algorithm step 1 in remarks.
`BarPast`	Numeric	See algorithm step 4 in remarks.
`oTLHiStartPr`	Numeric	Outputs the start price of current pennant's hi-line.
`oTLHiEndPr`	Numeric	Outputs the end price of current pennant's hi-line (this continues to be updated for BarsPast bars after the pennant is confirmed).
`oTLLoStartPr`	Numeric	Outputs the start price of current pennant's lo-line.
`oTLLoEndPr`	Numeric	Outputs the end price of current pennant's lo-line (this continues to be updated for BarsPast bars after the pennant is confirmed).

Remarks

If a new pennant is identified within BarsPast bars of the previous pennant, the new pennant supersedes the previous pennant.

For the purposes of this study, a pennant is defined as a converging consolidation pattern on a bar chart. This includes cases where the two straight lines bounding the pattern's highs and lows are sloping in opposite directions as well as in the same direction, as long as they are converging. The limiting case of when the two lines are parallel is included. Note that this definition automatically includes certain consolidation patterns that may otherwise be known as "flags".

Algorithm

1. Check if the Consolidation Index (True Price Channel/Average True Range) is low enough (i.e., < MaxConsolIndex) to indicate a significant degree of price consolidation. ConsolIndex values can range between 1 and Length. Low values indicate price consolidation, high values indicate price extension. To identify reasonable consolidation patterns, some suitable combinations of Length/ MaxConsolIndex could be 7/1.5 and 15/2. (Also see the RS_PriceExtension function, which looks for high values of ConsolIndex.)
2. If price consolidation is confirmed, outline the pattern by "drawing" a linear regression line through the pattern's highs and a second such line through the pattern's lows, and check if the lines are converging.
3. If the lines are converging, a pennant exists; finish defining the pennant by shifting the lines outwards as much as necessary to fully enclose all the bars in the consolidation pattern.
4. For the next BarsPast bars, or until the pennant converges, whichever comes first, look for the price to breakout of the pennant, up or down, whichever comes first. (Simultaneous up and down breakouts are ignored.)

Example

```
inputs: Length(7), MaxConsolIndex(1.5), BarsPast(5) ;
variables: oTLHiStartPr(0), oTLHiEndPr(0), oTLLoStartPr(0), oTLLoEndPr(0) ;
Value1 = Pennant(Length, MaxConsolIndex, BarsPast, oTLHiStartPr, oTLHiEndPr,
oTLLoStartPr, oTLLoEndPr) ;
Plot1(Value1) ;
```

PercentChange (Function)

Disclaimer

The PercentChange function calculates the percent change in price of the current bar over the price length bars ago.

Syntax

```
PercentChange(Price, Length)
```

Returns (Double)

A numeric value containing the percent change in price of the current bar over the price length bars ago.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to be considered.
Length	Numeric	Sets the number of bars to be considered.

Remarks

PercentChange takes the difference in between the current value of Price and the Price specified by Length bars ago, and then divides the result by the Price of Length bars ago.

Example

```
PercentChange(Close, 14)
```

If the above returns a value of 0.35, there was a 35% increase in the closing price of the current bar compared to the closing price of 14 bars ago.

```
PercentChange(High, 25);
```

returns a value of -0.15 indicating that there was a 15% decrease in the High price of the current bar compared to the High price of 25 bars ago.

Additional Example

You could use the PercentChange function to alert you when the closing price has increased 10% from the closing price of 10 bars ago by using the following syntax:

```
Value1 = PercentChange(Close, 10);

Plot1(Value1, "PrctChange");

If Value1 >= .10 Then
 Alert;
```

Percentile (Function)

Disclaimer

The `Percentile` function returns a price value at the boundary of the specified percentile level.

Syntax

```
Percentile(PcntRank, Price, Length)
```

Returns (Double)

A numeric value containing the percentile (k-th value) of a specified period, as defined by `Price` and `Length`.

Parameters

Name	Type	Description
`PcntRank`	Numeric	Sets the percentile to be considered using a decimal value between 0 and 1. Enter .25 for %25.
`Price`	Numeric	Specifies which bar value (price, function, or formula) to be considered.
`Length`	Numeric	Sets the number of bars to be considered.

Remarks

The Percentile value that is returned is at the point in the `Price` data series where a specified percentage of the prices are below the `PcntRank` level.

Example

Assigns to `Value1` the value of the 25th percentile, based on the Closing prices of the last 10 bars:

```
Value1 = Percentile(.25, Close, 10);
```

Assigns to `Value2` the value of the 50th percentile, based on the Range of the last 21 bars:

```
Value2 = Percentile(.5, Range, 21);
```

PercentileArray (Function)

Disclaimer

The PercentileArray function returns the value in an array that corresponds to the PctRank input..

Syntax

```
PercentileArray(PctRank, PriceArray, Size)
```

Parameters

Name	Type	Description
PctRank	Numeric	Sets the percentile value to use. Enter .25 for %25.
PriceArray	Numeric Array	Specifies the name of a numeric array containing values upon which the average deviation is calculated.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.

Returns

A numeric value containing the percentile (k-th value) of a specified period, as defined by Price and Length. The Function will return a –1 if the Size is greater than the referenced array.

Usage

Calculates the Percentile (k-th value) of a specified period.

The percentile is a value within the bounds of the array, such that PctRank percent of the observations are less than the PctRank value, and (1 - PctRank) percent of the observations are greater than the PctRank value.

Remarks

AvgDeviation is a measure of the variability in a data set.

The AvgDeviationArray function first calculates a standard average (mean) of the specified PriceArray using the Length input parameter for the number of trailing bars. Next it measures the absolute difference (ABSValue, which removes the negative sign) of each array data element from the mean and averages the sum of the differences.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The AvgDeviationArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Assigns to Value1 the 10 bar average deviation from the mean for a specified array.

```
Array: myArray[10](0);
{… (assign values to array) }
Value1 = AvgDeviationArray (myArray, 10);
```

PercentR (Function)

Disclaimer

The PercentR function evaluates the price range over Length number of bars. It then returns a percentage of where the current price lies, related to the evaluated trading range.

Syntax

```
PercentR(Length)
```

Returns (Double)

A numeric value containing the current %R value.

Parameters

Name	Type	Description
Length	Numeric	Sets the number of bars to consider in the calculation of PercentR.

Remarks

The formula used to calculate the Percentage Range follows:

$\%R=100*((H_r-C)/(H_r-L_r))$

…where

r = The time period selected

Hr = The highest High of that period

Lr = The lowest Low of that period

C = Today's Close

The input Length can be hard coded with a whole number or can be replaced by a numeric simple input whose default value is a whole number.

Example

Plots the PercentR over a length of 10 bars.

```
Plot1(PercentR(10);
```

Reference

Williams, Larry. *How I Made One Million Dollars Last Year Trading Commodities.* Monterey, California: Conceptual Management, 1973 2nd edition.

To be consistent with other oscillators, PercentR takes William's value and subtracts it from one hundred (100). For example, if William's Percent R returns 80, PercentR will return 20.

PercentRank (Function)

Disclaimer

The PercentRank function calculates the percentile rank of a bar value in a data set.

Syntax

```
PercentRank(PriceToRank, Price, Length)
```

Returns (Integer)

The percentile rank of a value in a data set. The function returns -1 if PriceToRank is outside the range of the data set or if there is a Length error.

Parameters

Name	Type	Description
PriceToRank	Numeric	Sets the value to be ranked within the Price data series.
Price	Numeric	Specifies which bar value (price, function, or formula) to be considered.
Length	Numeric	Sets the number of bars to be considered.

Remarks

This function can be used to evaluate the relative standing of a value within a data set, as specified by the Price and Length inputs. If PriceToRank < the lowest price in the data series, PriceToRank > the highest price in the data series or the Length <= 0 the function will return -1.

Example

Assigns to Value1 the percentage rank of the value 260 within the data set defined by the Closes of the last 21 bars:

```
Value1 = PercentRank(260, Close, 21);
```

Assigns to Value2 the percentage rank of the value 3 within the data set defined by the Ranges of the last 14 bars:

```
Value2 = PercentRank(3, Close, 14);
```

PercentRankArray (Function)

Disclaimer

The PercentRankArray function returns the rank of a value within an array as a percentage of the data set.

Syntax

```
PercentRankArray(PriceToRank, PriceArray, Size)
```

Returns (Double)

A numeric value the rank of a value in a data set as a percentage of the data set. The function will return -1 if the `Size` is greater than the referenced Array.

Parameters

Name	Type	Description
`PriceToRank`	Numeric	Sets the value to rank within the specified array.
`PriceArray`	Numeric Array	Specifies the name of a numeric array containing values upon which the rank iis calculated.
`Size`	Numeric	Sets the number of data elements (size) in the array to evaluate.

Remarks

This function can be used to evaluate the relative standing of a value within an array.

The value for the `Size` input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The `PercentRankArray` function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Assigns to `Value1` the rank of a price value of 45 in a user defined array.

```
Array: myArray[30](0);
{… (assign values to array) }
Value1 = PercentRankArray(45,myArray, 30);
```

Permutation (Function)

Disclaimer

The Permutation function calculates the number of permutations for a given number of objects that can be selected from a range of objects.

Syntax

```
Permutation(Num, NumChosen)
```

Returns (Double)

A numeric value containing the number of permutations for a given number (NumChosen) of objects that can be selected from number objects (Num).

Parameters

Name	Type	Description
Num	Numeric	Sets the number of bars to be considered.
NumChosen	Numeric	Sets the number of objects within the range (Num) that can be selected.

Usage

A permutation is any set or subset of objects or events where internal order is significant. If Num <= 0, NumChosen < 0, or Num < NumChosen the Function will return a -1. -1 will also be returned if there is a divisor of zero within the calculation.

Example

Assigns to Value1 the number of permutations for a range of 50, containing permutations with 3 elements:

```
Value1 = Permutation(50, 3);
```

Pivot (Function)

Disclaimer

The Pivot function returns the value of a pivot point and the number of bars ago the pivot occurred.

Syntax

```
Pivot(Price, Length, LeftStrength, RightStrength, Instance, HiLo, oPivotPrice,
oPivotBar)
```

Returns (Integer)

The oPivotPrice and oPivotBar output parameters return the price value of the pivot point and the number of bars ago it occurred. The Pivot function itself returns a value of 1 if a pivot is found, and -1 if not found.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to be considered.
Length	Numeric	Sets the number of bars to consider for the pivot.
LeftStrength	Numeric	Sets the required number of bars on the left side of the pivot bar.
RightStrength	Numeric	Sets the required number of bars on the right side of the pivot bar.
Instance	Numeric	Sets which occurrence (that is, 1 = most recent, 2 = 2nd most recent, and so on) to return.
HiLo	Numeric	Sets which pivot values to return, 1 = High, -1 = Low.
oPivotPrice	Numeric	Outputs the specified bar value at the pivot point.
oPivotBar	Numeric	Outputs the number of bars ago the pivot point occurred.

Remarks

This function is used to find pivot or swing points. This function allows you to specify variable strength sides, as well as whether the pivot point should be a high or low pivot.

Example

Assigns to Value2 the most recent pivot low price over the last 21 bars and to Value3 the number of bars ago the pivot low occurred using a left strength of 4 and a right strength of 2.

```
Vars: oPivotPrice(0), oPivotBar(0);

Value1 = Pivot(Low,21,4,2,1,-1,oPivotPrice,oPivotBar);

Value2 = oPivotPrice;

Value3 = oPivotbar;
```

PivotHighVS (Function)

Disclaimer

The PivotHighVS function returns the value of the specified instance of a High Pivot bar with variable strength sides.

Syntax

```
PivotHighVS(Instance, Price, LeftStrength, RightStrength, Length)
```

Returns (Double)

A numeric value containing the price of the high pivot bar, with variable strength sides. A value of -1 is returned if there is no high pivot, within the length specified.

Parameters

Name	Type	Description
Instance	Numeric	Sets which occurrence (that is, 1 = most recent, 2 = 2nd most recent, and so on) to return.
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
LeftStrength	Numeric	Sets the required number of bars on left side of the pivot bar.
RightStrength	Numeric	Sets the required number of bars on right side of the pivot bar.
Length	Numeric	Sets the number of bars to be considered.

Example

Assigns to Value1 the most recently occurring high pivot, in the last 21 bars, that has a left side strength of 4 and a right side strength of 2:

```
Value1 = PivotHighVS(1, High, 4, 2, 21);
```

Assigns to Value2 the value of the second most recently occurring high pivot, in the last 10 bars, that has a left side strength of 3 and a right side strength of 2:

```
Value2 = PivotHighVS(2, High, 3, 2, 10);
```

Additional Example

If you wanted to place a mark on the High of the high pivot bar, with a left strength of 4 and a right strength of 2, you could use the following syntax:

```
If PivotHighVS(1, High, 4, 2, 3) <> -1 Then
 Plot1[2](High[2], "PivotH");
```

PivotHighVSBar (Function)

Disclaimer

The PivotHighVSBar function returns the number of bars ago that a specified instance of a high pivot bar, with variable strength sides, occurred.

Syntax

```
PivotHighVSBar(Instance, Price, LeftStrength, RightStrength, Length)
```

Returns (Integer)

A numeric value containing the number of bars ago that a specific instance of a high pivot bar occurred. A value of -1 is returned if there is no high pivot, within the length specified.

Parameters

Name	**Type**	**Description**
Instance	Numeric	Sets which occurrence (that is, 1 = most recent, 2 = 2nd most recent, and so on) to return.
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
LeftStrength	Numeric	Sets the required number of bars on left side of the pivot bar.
RightStrength	Numeric	Sets the required number of bars on right side of the pivot bar.
Length	Numeric	Sets the number of bars to be considered.

Example

Assigns to Value1 the number of bars ago that the most recently occurring high pivot, in the last 21 bars, that has a left-side strength of 4 and a right-side strength of 2, occurred.:

```
Value1 = PivotHighVSBar(1, High, 4, 2, 21);
```

Assigns to Value2 the number of bars ago that the second most recently occurring high pivot, in the last 10 bars, that has a left side strength of 3 and a right side strength of 2, occurred:

```
Value2 = PivotHighVSBar(2, High, 3, 2, 10);
```

Additional Example

If you wanted to place a mark on the High of the high pivot bar, with a left strength of 4 and a right strength of 2, you could use the following syntax:

```
If PivotHighVSBar(1, High, 4, 2, 3) = 2 Then
 Plot1[2](High[2], "PivotH");
```

PivotLowVS (Function)

Disclaimer

The PivotLowVS function returns the value of the specified instance of a low pivot bar with variable strength sides.

Syntax

```
PivotLowVS(Instance, Price, LeftStrength, RightStrength, Length)
```

Returns (Double)

A numeric value containing the price of the low pivot bar, with variable strength sides. A value of -1 is returned if there is no low pivot, within the length specified.

Parameters

Name	**Type**	**Description**
Instance	Numeric	Sets which occurrence (that is, 1 = most recent, 2 = 2nd most recent, and so on) to return.
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
LeftStrength	Numeric	Sets the required number of bars on left side of the pivot bar.
RightStrength	Numeric	Sets the required number of bars on right side of the pivot bar.
Length	Numeric	Sets the number of bars to be considered.

Examples

Assigns to Value1 the most recently occurring low pivot, in the last 21 bars, that has a left side strength of 4 and a right side strength of 2:

```
Value1 = PivotLowVS(1, Low, 4, 2, 21);
```

Assigns to Value2 the second most recently occurring low pivot, in the last 10 bars, that has a left side strength of 3 and a right side strength of 2:

```
Value2 = PivotLowVS(2, Low, 3, 2, 10);
```

Additional Example

If you wanted to place a mark on the Low of the low pivot bar, with a left strength of 4 and a right strength of 2, you could use the following syntax:

```
If PivotLowVS(1, Low, 4, 2, 3) <> -1 Then
 Plot1[2](Low[2], "PivotL");
```

PivotLowVSBar (Function)

Disclaimer

The PivotLowVSBar function returns the number of bars ago that a specified instance of a low pivot bar, with variable strength sides, occurred.

Syntax

```
PivotLowVSBar(Instance, Price, LeftStrength, RightStrength, Length)
```

Returns (Integer)

A numeric value containing the number of bars ago that a specific instance of a low pivot bar occurred. A value of -1 is returned if there is no low pivot, within the length specified.

Parameters

Name	Type	Description
Instance	Numeric	Sets which occurrence (that is, 1 = most recent, 2 = 2nd most recent, and so on) to return.
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
LeftStrength	Numeric	Sets the required number of bars on left side of the pivot bar.
RightStrength	Numeric	Sets the required number of bars on right side of the pivot bar.
Length	Numeric	Sets the number of bars to be considered.

Example

Assigns to Value1 the number of bars ago that the most recently occurring low pivot, in the last 21 bars, that has a left side strength of 4 and a right side strength of 2, occurred:

```
Value1 = PivotLowVSBar(1, Low, 4, 2, 21);
```

Assigns to Value2 the number of bars ago that the second most recently occurring low pivot, in the last 10 bars, that has a left side strength of 3 and a right side strength of 2, occurred:

```
Value2 = PivotLowVSBar(2, Low, 3, 2, 10);
```

Additional Example

If you wanted to place a mark on the Low of the low pivot bar, with a left strength of 4 and a right strength of 2, you could use the following syntax:

```
If PivotLowVSBar(1, Low, 4, 2, 3) = 2 Then
 Plot1[2](Low[2], "PivotL");
```

PivotReversalGen (Function)

Disclaimer

The PivotReversalGen series function provides a generalized approach to identifying pivot reversals.

Usage

```
PivotReversalGen(MinRStren, MaxRStren, LRFactor, HiLo, DrawLines, LinesColor,
oPivotPrice, oPivotRStren, oPivotLStren)
```

Returns (Integer)

The oPivotPrice, oPivotRStren,and oPivotLStren output parameters return the price value of the pivot point along with the right and left strength values from the pivot. The PivotReversalGen function itself returns a value of 1 if a pivot is found, and -1 if not found.

A numeric value of 1 in returned if the specified reversal pattern was found, 0 if not found.

Parameters

Name	Type	Description
MinRStren	Numeric	Sets the minimum right strength range value.
MaxRStren	Numeric	Sets the maximum right strength range value.
LRFactor	Numeric	Sets the multiplier used to calculate the left strength based on the right strength.
HiLo	Numeric	Sets the type of reversal to identify. 1=high pivot, -1=low pivot.
DrawLines	Boolean	Sets whether a horizontal line should be drawn from previous pivot lows to the current bar. True=draw line, False=no line.
LinesColor	Numeric	Sets the color of horizontal marker lines.
oPivotPrice	Numeric	Outputs the actual price value of the pivot bar.
oPivotRStren	Numeric	Outputs the right strength value of the pivot bar.
oPivotLStren	Numeric	Outputs the left strength value of the pivot bar.

Remarks

The function identifies when any price on the current bar breaches any previously un-breached "significant" high or low pivot reversal. If the current bar breaches more than one previously unbreached "significant" high pivots, the outputs pertain to the first pivot breached. The optional pointers, however, are drawn from each pivot high to the current bar.

A pivot is considered "significant" if its right strength is within the range specified by the MinRStren and MaxRStren inputs, and its left strength is greater than LRFactor times its right strength. If LRFactor is set to 0, the pivot's left strength is required to be greater than or equal to MinRStren.

The approach used in this function may also provide a more robust alternative to analysis techniques that look for "new highs" or "new lows", since even there, the real event of interest may be the breaching of a previously un-breached significant high or low point on the chart.

Example

Assigns output values for the pivot price, right strength, and left strength if a low pivot is found based on the minimum strength inputs provided and then draws a magenta, horiztontal from the previous pivot.

```
Vars: oPivotPrice(0),oPivotRStren(0),oPivotLStren(0);
Value1 = PivotReversalGen(1,30,1,-
1,True,Magenta,oPivotPrice,oPivotRStren,oPivotLStren);
if Value1 = 1 then
 begin
   Value2 = oPivotPrice;
   Value3 = oPivotRStren;
   Value4 = oPivotLStren;
 end;
```

PlaceOrder (Function)

Disclaimer

The PlaceOrder function is a low level system function that supports higher level 'wrapper' functions and is used to generate a string of parameters that is then makes a call to the order entry macro .PlaceOrder. In addition, the PlaceOrder function provides built-in runtime error checking and reporting.

NOTE The PlaceOrder function is only intended for use by designated TradeStation order functions listed under **See Also** below. Use these designated order functions in your EasyLanguage code to generate the appropriate order.

Syntax

```
PlaceOrder(OrderName, Frequency, Account, Action, SymbolCategory, MySymbol, Quantity,
Duration, GTDDate, Route, AllOrNone, ByMinusSellPlus, Discretionary, ECNSweep,
IfTouched, LimitPrice, NonDisplay, Peg, ShowOnly, StopPrice, TrailingAmount,
TrailingType);
```

Returns

PlaceOrder returns 1 if called on a "real-time" tick and if a trade is allowed based on the user input "Frequency". The function returns -1 (negative one) in other cases; for example, if the function is called on an historical bar. If there are errors in the order parameters a runtime error message will be generated and the order will not be placed.

See Also

LimitOrder, LimitIfTouchedOrder, MarketOrder, MarketIfTouchOrder, StopLimitOrder, StopMarketOrder, TrailingStopOrder

PositionProfitCustom (Function)

Disclaimer

The PositionProfitCustom function calculates a customized position profit value based on a specified price, for either current position profit or maximum position profit.

Syntax

```
PositionProfitCustom(LongProfit, ShortProfit, MaxPosProfit)
```

Returns (Double)

A customized position profit value based on a specified price, for either current position profit or maximum position profit.

Parameters

LongProfit	The price upon which the long-side profit calculation is based. Usually the High or Close of the bar is used.
ShortProfit	The price upon which the short-side profit calculation is based. Usually the Low or Close of the bar is used.
MaxPosProfit	A True/False value where, if it is set to True, the function calculates the Maximum Position Profit. If it is set to False, the function calculates the current position profit.

Remarks

This function allows you to specify the price value upon which the profit calculation is based. When there is no position, the function will return the position profit value, based on the most recent position.

Examples

Assigns to Value1 the Maximum Position Profit calculated from the High on the long side and from the Low on the short side:

```
Value1 = PositionProfitCustom(High, Low, True);
```

Assigns to Value2 the current Position Profit calculated from the Close for both the long and short sides:

```
Value2 = PositionProfitCustom(Close, Close, False);
```

Additional Example

If you wanted to place a long and a short exit on the Close when the maximum position profit, based on the extreme values (High and Low), reached $500, you could use the following syntax:

```
If PositionProfitCustom(High, Low, True) >= 500 AND
 MarketPosition <> 0 Then Begin
  Sell This Bar on Close;
  BuyToCover This Bar on Close;
End;
```

PriceOscillator (Function)

Disclaimer

The PriceOscillator series function calculates the difference between the Slow Moving Average and the Fast Moving Average.

Syntax

```
PriceOscillator(Price, FastLength, SlowLength)
```

Returns (Double)

The difference between the Fast Moving Average and the Slow Moving Average.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to consider.
FastLength	Numeric	Sets the number of bars over which the fast average will be calculated.
SlowLength	Numeric	Sets the number of bars over which the slow average will be calculated.

Example

Assigns to Value1 the difference between the 9 bar moving average and the 18 bar moving average, both based on the Close:

```
Value1 = PriceOscillator(Close, 9, 18);
```

Assigns to Value2 the difference between the 3 bar moving average and the 5 bar moving average, both based on the Close:

```
Value2 = PriceOscillator(Close, 3, 5);
```

Additional Example

If you wanted an Alert when the PriceOscillator, based on 10 and 20 bar moving averages, crosses above zero, indicating that the fast average has crossed above the slow average, you could use the following syntax:

```
If PriceOscillator(Close, 10, 20) Crosses Above 0 Then
 Alert("The Price Oscillator has crosses above zero");
```

PriceVolTrend (Function)

Disclaimer

The PriceVolTrend (PVT) function calculates the price volume trend for the current bar.

Syntax

```
PriceVolTrend
```

Returns (Double)

A numeric value containing PVT for the current bar.

Parameters

None

Remarks

PriceVolTrend multiplies the day's trade volume by the percentage difference between today's Close and yesterday's Close. It accumulates the resulting value for each day, either up or down.

The function is calculated as follows:

PriceVolTrend = (((C - C[1]) / C[1]) * V)

… for the first day, and as follows for subsequent days:

PriceVolTrend = (((C - C[1]) / C[1]) * V) + PriceVolTrend[1];

Example

```
Plot1(PriceVolTrend) ;
```

ProbAbove (Function)

Disclaimer

The ProbAbove function returns the probability that price will rise or remain above a price target. This function is designed primarily for use with Probability Maps.

Syntax

```
ProbAbove(PriceTarget, CurrentPrice, VoltyVal, BarsToGo)
```

Parameters

Name	Type	Description
PriceTarget	Numeric	Sets a future price value.
CurrentPrice	Numeric	Sets the price to be tested against the target price.
VoltyVal	Numeric	Sets the annualized volatility of the symbol being tested.
BarsToGo	Numeric	Sets the number of bars in the future to consider when evaluating the probability of being above PriceTarget.

Returns (Double)

A numeric value containing the probability that price will rise or remain above PriceTarget.

Remarks

ProbAbove calculates the probability of price being above some value at a particular point in the future, given the current price, volatility, and time remaining (in bars).

Example

Assigns to Value1 the chance that the symbol, which is currently trading at 100, will be trading at 110 or higher in exactly 30 bars into the future. The following returns **.**253 (represent 25.3%):

```
   Value1 = ProbAbove(110, 100, .50, 30);
```

The above example returns a value of .253 (representing a 25.3% probability).

ProbBelow (Function)

Disclaimer

The ProbAbove function returns the probability that price will fall or remain below a price target. This function is designed primarily for use with Probability Maps.

Syntax

```
ProbBelow(PriceTarget, CurrentPrice, VoltyVal, BarsToGo)
```

Parameters

Name	Type	Description
PriceTarget	Numeric	Sets a future price value.
CurrentPrice	Numeric	Sets the price to be tested against the target price.
VoltyVal	Numeric	Sets the annualized volatility of the symbol being tested.
BarsToGo	Numeric	Sets the number of bars in the future to consider when evaluating the probability of being above PriceTarget.

Returns (Double)

A numeric value containing the probability that price will fall or remain below PriceTarget.

Remarks

ProbBelow calculates the probability of price being below some value at a particular point in the future, given the current price, volatility, and time remaining (in bars).

Example

Assigns to Value1 the chance that the symbol, which is currently trading at 110, will be trading at 100 or lower in exactly 30 bars into the future. Value1 will be set to .253:

```
Value1 = ProbBelow(100, 110, .50, 30);
```

The above example returns a value of .253 (representing a 25.3% probability).

ProbBetween (Function)

Disclaimer

The ProbBetween function returns the probability that price will be within the range specified by Low and High target prices. This function is designed primarily for use with Probability Maps.

Syntax

```
ProbBetween(LowTarget, HighTarget, CurrentPrice, VoltyVal, BarsToGo)
```

Parameters

Name	Type	Description
LowTarget	Numeric	Sets the low-end price target.
HighTarget	Numeric	Sets the high-end price target..
CurrentPrice	Numeric	Sets the price to be tested against the target price.
VoltyVal	Numeric	Sets the annualized volatility of the symbol being tested.
BarsToGo	Numeric	Sets the number of bars in the future to consider when evaluating the probability of being between the price targets.

Returns (Double)

A numeric value containing the probability that price will be within the range specified by LowTarget and HighTarget.

Remarks

ProbBetween is used to determine the likelihood of price staying within a range. Conversely, if Probability Between returns 25%, it can be concluded that there is a 75% chance that price will move outside of the specified range on the specified future date.

ProbBetween calculates the probability of price being within a specified price range at a particular point in the future, given the current price, volatility and time remaining in bars.

Example

Assigns to Value1 the chance that the symbol, which is currently trading at 105, will be trading between 100 and 110 in exactly 30 bars into the future:

```
Value1 = ProbBetween(110, 100, 105, .50, 30);
```

The above example returns a value of .2605 (representing a 25.05% probability).

Quartile (Function)

Disclaimer

The Quartile function returns the price value at the specified percentile.

Syntax

```
Quartile(QRank, Price, Length)
```

Returns (Double)

A numeric value containing the quartile price at the specified ranking level. The function returns -1 if QRank is less than 0 or greater than 4.

Parameters

Name	Type	Description
QRank	Numeric	Sets the percentile ranking level. Enter 0 for 0%, 1 for 25%, 2 for 50%, 3 for 75%, or 4 for 100%.
Price	Numeric	Specifies which bar value (price, function, or formula) to be considered.
Length	Numeric	Sets the number of bars to be considered.

Remarks

Quartile is a measure of "non-central" location used to split ordered data into four quarters.

The four quartiles can be defined as follow:

Q1: a value such that 25% of the observations are smaller, and 75% of the observations are larger.

Q2: a value such that 50% of the observations are smaller, and 50% of the observations are larger.

Q3: a value such that 75% of the observations are smaller, and 25% of the observations are larger.

Q4: a value such that 100% of the observations are smaller, and none of the observations are larger.

Example

Assigns to Value1 the first quartile value at 25% of the observations are smaller, and 75% of the observations are larger, based on the ordered Highs from the last 20 bars:

```
Value1 = Quartile(1, High, 20);
```

Assigns to Value2 the third quartile value at which 75% of the observations are smaller and 25% of the observations are larger, based on the ordered Closes from the last 10 bars:

```
Value2 = Quartile(3, Close, 10);
```

QuartileArray (Function)

Disclaimer

The quartile of a data set stored in an array.

Note All array-based function calculations begin with array element 1.

Function

```
QuartileArray(QRank, PriceArray, Size)
```

Parameters

`QRank`	A numeric expression that determines which "`QRank`" is being referenced. This Input can exclusively be an integer from 0 to 4.
`PriceArray`	A numeric array upon which the Quartile calculation is performed.
`Size`	A numeric expression representing the number of elements in the original array that have been used.

Returns

A numeric value containing the quartile of a data set. If the `QRank` Input is less than 0 or the `QRank` Input is greater than 4, the function returns a -1. Also, the Function will return a -1 if the `Size` is greater than the referenced array.

Usage

`QuartileArray` is a measure of "non-central" location used to split ordered data into four quarters.

The four quartiles can be defined as follow:

Q1: a value such that 25% of the observations are smaller, and 75% of the observations are larger.

Q2: a value such that 50% of the observations are smaller, and 50% of the observations are larger.

Q3: a value such that 75% of the observations are smaller, and 25% of the observations are larger.

Q4: a value such that 100% of the observations are smaller, and none of the observations are larger.

Range (Function)

Disclaimer

The Range function subtracts the Low of a bar from the High to determine the range of the bar.

Syntax

```
Range
```

Returns (Double)

A numeric value containing the current range (High minus Low) value.

Parameters

None

Example

```
Plot1(Range);
```

RangeLeader (Function)

Disclaimer

The `RangeLeader` function checks to see if the current bar is a range leader.

Syntax

```
RangeLeader
```

Returns (Integer)

A value of 1 is returned if the current bar is considered a range leader based on a pair of conditions (see Remarks). `0` if it is not.

Parameters

None

Remarks

This function compares the current and previous bar and looks at two conditions:

- Whether the mid-point of the current bar is greater than the previous `High` or less than the previous `Low.`
- Whether the `Range` of the current bar is greater than the `Range` of the previous bar. If both conditions are met, the function returns 1; if one or neither is met, the function returns 0.

Example

```
Value1 = RangeLeader;
```

RateOfChange (Function)

Disclaimer

The RateOfChange function returns the Rate of Change calculation for the current bar.

Syntax

```
RateOfChange(Price, Length)
```

Returns (Double)

A numeric value containing ROC for the current bar.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to be considered.
Length	Numeric	Sets the number of bars to be considered.

Remarks

RateOfChange is calculated using the division method. The equivalent of the subtraction method is found in the function Momentum.

ROC = ((Price/Price$_p$) -1)*100

…where

Price = Current value (that is, Close, High, Low, and so on)

Price$_p$= Previous value (determined by the value returned for the input Length)

The input Price, is usually hard coded with some bar attribute such as Close, Open, High, Low, and Volume or is replaced with a numeric series type input. However, it can be replaced with a valid EasyLanguage expression. For example: Close + Open, or Average(RSI(Close,14),14).

Example

```
Value1 = RateOfChange(Close,20);
```

Reference

P.J. Kaufman, *The New Commodity Trading Systems & Methods*.

RecentOcc (Function)

Disclaimer

The RecentOcc function returns the number of bars ago a specified expression was True.

Syntax

```
RecentOcc(Test, Length, Instance, MLFlag)
```

Returns (Integer)

A numeric value containing the number of bars ago that the specified Test was True; -1 if Test was not found to be True within the last Length bars.

Parameters

Name	Type	Description
Test	TrueFalse	Sets the true/false expression to check for (that is, Close > Open).
Length	Numeric	Sets the number of bars to check.
Instance	Numeric	Sets which occurrence, for example, 1 = most recent, 2 = 2nd most recent, and so on.
MLFlag	Numeric	Sets which where to start searching, 1 = Most Recent, -1 = Least Recent

Remarks

The RecentOcc function is designed to identify when the 'Nth' Instance of Test occurred. MLFlag Is used to determine if the function begins searching for the most recent occurrence or the least recent occurrence. Length specifies the number of bars to check for the condition.

The function always returns a number representing how many bars ago the true/false expression was fulfilled (0 = current bar, 1 = one bar ago, 2 = 2 bars ago …). If the function does not find a bar within the specified Length that meets the criteria, it will return a -1.

Note When using RecentOcc in an analysis technique, always check the result of the function before using the value in another calculation. If the condition occurred over Length number of bars, a -1 value will be returned.

Example

```
Value1 = RecentOcc(Close>0, 5, 1, 1);
```

See Also,

MRO, LRO

Round2Fraction (Function)

Disclaimer

The `Round2Fraction` function calculates a value rounded to the nearest minmove fraction.

Syntax

```
Round2Fraction(DecAmt)
```

Parameters

Name	Type	Description
`DecAmt`	Numeric	Sets a decimal value to be rounded to the nearest fractional price scale.

Returns (Double)

A numeric value containing the nearest fractional value for a decimal variable.

Remarks

The nearest fraction is based on the price scale for the symbol type being evaluated. You can determine the price scale for any symbol by looking at the symbol properties.

Example

Assigns to `Value1` the number 100.375 as the nearest fractional value for a stock with a price scale of 1/8th.

```
Value1 = RoundToFraction(100.3732);
```

RS_Average (Function)

Disclaimer

The RS_Average function has been designed for use with intraday charts only, and *must* be called in conjunction with the RS_DailyDataArray function. The RS_DailyDataArray function extracts daily data from intraday data and makes it available to subsequently called functions like RS_Average that use that daily data.

This function returns the simple average of the most recent NumDays elements in the RowToAvg row of the DataArray.

Syntax

```
RS_Average(RowToAvg, NumDays, Offset, DataArray, Index)
```

Returns (Double)

The simple average of the most recent NumDays elements in the RowToAvg row within the DataArray. Note that DataArray is typically an output parameter of the RS_DailyDataArray function.

Parameters

Name	Type	Description
RowToAvg	Numeric	Specifies the row of the DataArray that contains the daily data to be averaged. Should be a whole number in the 1-to-12 range.
NumDays	Numeric	The number of days to be used in the function calculation. Should be <= MaxNumDays (see DataArray parameter below).
Offset	Numeric	This should be a non-negative whole number. If this is 0, the most recent NumDays, not including the current day, will be used; if this is 1, the most recent NumDays, not counting the current day or the previous day, will be used; and so on. (Note: Make sure to pass in NumDays + Max Offset into the NumDays input for the RS_DailyDataArray function.)
DataArray	Numeric	An array that contains the daily data to be used. This is an output of the RS_DailyDataArray function, and in input to the RS_Average function. (Note: The calling routine must declare the size of this array as 12-by-MaxNumDays, with MaxNumDays >= NumDays + Max Offset.)
Index	Numeric	The number of the DataArray column in which the data for the current day is being loaded. This is an output of the RS_DailyDataArray function, and an input to the RS_Average function.

Remarks

This approach is intended primarily for use with RadarScreen indicators, where only one data stream can be used; in charting, similar calculations can be performed more easily using a multi-data approach.

Example

```
inputs: NumDays(3), RowToAvg(6), Offset( 0 );
variables: NumDaysPlusOffset(NumDays + Offset), Index(0);
arrays: DataArray[12,100](0), SubArray[3](0);
Value1 = RS_DailyDataArray(NumDaysPlusOffset, DataArray, Index, SubArray);
if CurrentBar = 1 or Date <> Date[1] then
       Value2 = RS_Average(RowToAvg, NumDays, Offset, DataArray, Index);
Plot1(Value2) ;
```

RS_BarsPerDay (Function)

Disclaimer

The RS_BarsPerDay function returns an estimate of the number of bars in a typical day for the chart it is applied to.

Syntax

```
RS_BarsPerDay
```

Returns (Integer)

An estimate of the number of bars in a typical day for the chart it is applied to.

Parameters

None.

Remarks

The estimate returned by this function does not make an adjustment for empty bars. If a 24hr custom session is used, with 1-min bars, the estimate returned will be 1440 bars per day, even though many of those bars will typically be empty.

Example

```
Value1 = RS_BarsPerDay;
```

RS_DailyDataArray (Series Function)

Disclaimer

The RS-DailyDataArray series function has been designed for use with intraday charts only. It extracts daily data from intraday data and makes it available to subsequently called functions like RS_Average and RS_Extremes that use this daily data.

Syntax

```
RS_DailyDataArray(NumDays, oDataArray, oIndex, oSubArray)
```

Returns (Integer)

The oDataArray, oIndex, and oSubArray output parameters return the data items listed in the parameters table. The function itself returns a value of 1.

Parameters

Name	**Type**	**Description**
NumDays	Numeric	Number of days of daily data required. This is used to determine the initial look-back period, so that NumDays worth of data is available to CurrentBar = 1.
oDataArray	Numeric	Output array (2 dimensional) returns 12 daily data items for each of the preceding NumDays, and may contain data for additional days. (Note: The calling routine must declare the size of this array as 12-by-MaxNumDays, with MaxNumDays >= NumDays.) At least NumDays worth of data will be available at CurrentBar = 1. After that, additional days of data will continue to accumulate in the array until the array is filled up. The array is implemented as a circular buffer, and loads backwards starting from column MaxNumDays down to column 0. Each time a loading pass is completed, the array starts re-loading from column MaxNumDays again. The 12 daily data items are as follows: 1 – Day open 2 – Day high 3 – Day low 4 – Day close (not available for current day) 5 – Day volume 6 – Day midrange (not available for current day) 7 – Day true high (not available for current day) 8 – Day true low (not available for current day) 9 – Day true range (not available for current day) 10 – Day high bar number 11 – Day low bar number 12 – Day number
oIndex	Numeric	Output variable returns the number of the oDataArray column in which the data for the current day is being loaded.
oSubArray	Numeric	Output array returns 3 pieces of data – the Day high (1), Day low (2), and Day number (3) of the final partial day in the MaxBarsBack buffer, at the bar preceding CurrentBar = 1. (Note: The calling routine must declare the size of this array to be 3; element 0 is not used.)

Remarks

This approach is intended primarily for use with RadarScreen indicators, where only one datastream can be used; in charting, similar calculations can be performed more easily using a multi-data approach.

This function will raise a run time error if it is applied to any chart other than intraday (minute bars).

Example

```
inputs: NumDays( 3 ), RowToAvg( 6 ), Offset( 0 ) ;
variables: NumDaysPlusOffset( NumDays + Offset ), Index( 0 ) ;
arrays: DataArray[ 12, 100 ]( 0 ), SubArray[3]( 0 ) ;
Value1 = RS_DailyDataArray( NumDaysPlusOffset, DataArray, Index, SubArray ) ;
Value2 = RS_Average( RowToAvg, NumDays, Offset, DataArray, Index ) ;
Plot1( Value2 ) ;
```

RS_Extremes (Function)

Disclaimer

This function has been designed for use with intraday charts only, and *must* be called in conjunction with the RS_DailyDataArray function. The RS_DailyDataArray function extracts daily data from intraday data and makes it available to subsequently called functions like RS_Extremes that use that daily data.

This function outputs the highest high/lowest low prices and relative day numbers of the most recent NumDays, not including the current day.

Usage

```
Value2 = RS_Extremes(NumDays, DataArray, Index, oPrevHighest, oPrevHighestDay,
oPrevLowest, oPrevLowestDay);
```

Returns (Integer)

This function has a dummy return of 1. All the useful information is passed back to the caller via the output parameters. For more information see Multiple Output Functions.

Parameters

NumDays	(Input) The number of days to be used in the function calculation. The most recent NumDays, not including the current day, will be used. NumDays should be <= MaxNumDays (see DataArray parameter below)..
DataArray	(Input) The array that contains the daily data to be used. This is an output of the RS_DailyDataArray function, and in input to the RS_Extremes function. (Note: The calling routine must declare the size of this array as 12-by-MaxNumDays, with MaxNumDays >= NumDays.)
Index	(Input) The number of the DataArray column in which the data for the current day is being loaded. This is an output of the RS_DailyDataArray function, and an input to the RS_Extremes function.
oPrevHighest	(Output) The array that contains the daily data to be used. This is an output of the RS_DailyDataArray function, and in input to the RS_Average function. (Note: The calling routine must declare the size of this array as 12-by-MaxNumDays, with MaxNumDays >= NumDays + Max Offset.)
oPrevHighestDay	(Output) The number of the DataArray column in which the data for the current day is being loaded. This is an output of the RS_DailyDataArray function, and an input to the RS_Average function.
oPrevLowest	(Output) The lowest low of the previous NumDays (i.e., not including the current day.)
oPrevLowestDay	(Output) The relative day number of the lowest low of the previous NumDays. If the LL was on the previous day (the most recent day included), this output is 0; if the LL was on the second previous day, this output is 1; and so on.

Remarks

This approach is intended primarily for use with RadarScreen indicators, where only one datastream can be used; in charting, similar calculations can be performed more easily using a multi-data approach.

Example

```
inputs: NumDays( 3 ) ;
variables: Index(0), oPrevHighest(0), oPrevHighestDay(0), oPrevLowest(0),
oPrevLowestDay(0) ;
```

```
arrays: DataArray[12, 100] (0),
 SubArray[3] (0) ;
Value1 = RS_DailyDataArray(NumDays, DataArray, Index, SubArray) ;
if CurrentBar = 1 or Date <> Date[1] then
Value2 = RS_Extremes(NumDays, DataArray, Index, oPrevHighest, oPrevHighestDay,
oPrevLowest, oPrevLowestDay ) ;
Plot1(oPrevHighest) ;
Plot2(oPrevLowest) ;
```

RS_PriceExtension (Function)

Disclaimer

This function has been designed for use with intraday charts only, and *must* be called in conjunction with the `RS_DailyDataArray` function. The latter extracts daily data from intraday data and makes it available to subsequently called functions like this one that use that daily data.

This function identifies price extensions, or strong, multiple-day moves. It does this by looking for high values of Consolidation Index (True Price Channel / Average True Range). ConsolIndex values can range between 1 and Length. Low values indicate price consolidation, high values indicate price extension.

(Also see the Pennant function, which looks for low values of ConsolIndex.)

Usage

```
Value3 = RS_PriceExtension(NumDays, MinConsolIndex, FinalRangeFactor, PrevTrHighest,
PrevTrLowest, PrevATR, DataArray, Index);
```

Returns

0: No extension found

1: Day following up extension

2: Day following down extension

Parameters (all inputs, no outputs)

NumDays	The number of days to be used in the function calculation. The most recent NumDays, not including the current day, will be used. NumDays should be <= MaxNumDays (see DataArray parameter below).
MinConsolIndex	This input should be in the 1-to-NumDays range; the larger the number, the more extended the price.
FinalRangeFactor	This factor ensures that the final day's move is good-sized - final day's TR will be >= oNumDaysATR * FinalRangeFactor.
PrevTrHighest	The highest true-high of the previous NumDays (i.e., not including the current day.) Typically, this would be obtained from a prior call to the `RS_TrueExtremes` function.
PrevTrLowest	The lowest true-low of the previous NumDays (i.e., not including the current day.) Typically, this would be obtained from a prior call to the `RS_TrueExtremes` function.
PrevATR	The average true range for the previous NumDays (i.e., not including the current day.) Typically, this would be obtained from a prior call to the `RS_TrueExtremes` function.
DataArray	The array that contains the daily data to be used. This is an output of the `RS_DailyDataArray` function, and in input to the `RS_PriceExtension` function. (Note: The calling routine must declare the size of this array as 12-by-MaxNumDays, with MaxNumDays >= NumDays.)
Index	The number of the DataArray column in which the data for the current day is being loaded. This is an output of the `RS_DailyDataArray` function, and an input to the `RS_PriceExtension` function.

Remarks

This approach is intended primarily for use with RadarScreen indicators, where only one datastream can be used; in charting, similar calculations can be performed more easily using a multi-data approach.

Example

```
inputs: NumDays( 3 ), MinConsolIndex( 2.25 ), FinalRangeFactor( 1 ) ;
variables: Index(0), PrevTrHighest(0), PrevTrLowest(0), PrevATR(0) ;
arrays: DataArray[ 12, 100 ] (0), SubArray[3] (0) ;
Value1 = RS_DailyDataArray(NumDays, DataArray, Index, SubArray) ;
if CurrentBar = 1 or Date <> Date[1] then
 begin
 Value2 = RS_TrueExtremes(NumDays, DataArray, Index, PrevTrHighest, PrevTrLowest, PrevATR) ;
            Value3 = RS_PriceExtension(NumDays, MinConsolIndex, FinalRangeFactor, PrevTrHighest,
            PrevTrLowest, PrevATR, DataArray, Index) ;
 end ;
Plot1(Value3) ;
```

RS_ReversalPatterns (Series Function)

Disclaimer

This function has been designed for use with intraday charts only, and *must* be called in conjunction with the RS_DailyDataArray function. The latter extracts daily data from intraday data and makes it available to subsequently called functions like this one that use that daily data.

This function outputs "setups" and "triggers" for the specified reversal direction and criteria, bar by bar (series function). Only the first trigger bar of the day is recognized - both setup and trigger are negated on the bar following the trigger bar.

Reversal Pattern Criteria

Crit1, Gap & Reverse (same day) - Market starts out very strong strong today, with a large opening CurrDayOpGap, but cannot sustain move and starts fading, triggering alert as it pulls back into yesterday's range.

Crit2, Falter & Reverse (2-day) - Market started out strong yesterday, but retracted by the end of the day, and continues the reversal today, triggering alert as it pulls out of yesterday's range.

Syntax

```
Value2 = RS_ReversalPatterns(RevDirection, RevCriteria, GapSizeFactor, DataArray,
Index, SubArray, oSetup, oTrigger) ;
```

Returns (Integer)

This function has a dummy return of 1. All the useful information is passed back to the caller via the output parameters. For more information see Multiple Output Functions.

Parameters

RevDirection	(Input) 1 for down reversal, 2 for up reversal.
RevCriteria	(Input) 1 for Crit1, 2 for Crit2.
GapSizeFactor	(Input) This factor is only used when RevCriteria = 1; it forces CurrDayOpGap to be >= PrevDayTR * GapSizeFactor.
DataArray	(Input) The array that contains the daily data to be used. This is an output of the RS_DailyDataArray function, and in input to the RS_ReversalPatterns function. (Note: The calling routine must declare the size of this array as 12-by-MaxNumDays, with MaxNumDays >= NumDays.)
Index	(Input) The number of the DataArray column in which the data for the current day is being loaded. This is an output of the RS_DailyDataArray function, and an input to the RS_ReversalPatterns function.
SubArray	(Input) This subordinate array contains some additional information for the final partial day in the MaxBarsBack buffer. This array is also an output of the RS_DailyDataArray function, and in input to the RS_ReversalPatterns function. (Note: The calling routine must declare the size of this array to be 3; element 0 is not used.)
oSetup	(Output) This is 1 if a setup exists, 0 otherwise.
oTrigger	(Output) At the bar the trigger condition is met, this output indicates the trigger price. At all other times this output is 0.

Remarks

Crit 2 above needs 2 days' history, so the NumDays input in the RS_DailyDataArray function should be set to at least 2 when Crit 2 is specified.

This approach is intended primarily for use with RadarScreen indicators, where only one datastream can be used; in charting, similar calculations can be performed more easily using a multi-data approach.

Example

```
inputs: NumDays(3), RevDirection(1), RevCriteria(1), GapSizeFactor(.2) ;
variables: Index(0), oSetup (0), oTrigger(0) ;
arrays: DataArray[12, 100] (0), SubArray[3] (0) ;
Value1 = RS_DailyDataArray( NumDays, DataArray, Index, SubArray ) ;
Value2 = RS_ReversalPatterns(RevDirection, RevCriteria, GapSizeFactor, DataArray, Index, SubArray, oSetup,
oTrigger) ;
Plot1(oSetup) ;
Plot2(oTrigger) ;
```

RS_TrueExtremes (Function)

Disclaimer

This function has been designed for use with intraday charts only, and *must* be called in conjunction with the RS_DailyDataArray function. The RS_DailyDataArray function extracts daily data from intraday data and makes it available to subsequently called functions like RS_TrueExtremes that use that daily data.

This function outputs the highest true-high/lowest true-low prices as well as the average true range of the most recent NumDays, not including the current day.

Usage

```
Value2 = RS_TrueExtremes(NumDays, DataArray, Index, oPrevTrHighest, oPrevTrLowest,
oPrevATR);
```

Returns (Integer)

This function has a dummy return of 1. All the useful information is passed back to the caller via the output parameters. For more information see Multiple Output Functions.

Parameters

NumDays	(Input) The number of days to be used in the function calculation. The most recent NumDays, not including the current day, will be used. NumDays should be <= MaxNumDays (see DataArray parameter below).
DataArray	(Input) The array that contains the daily data to be used. This is an output of the RS_DailyDataArray function, and in input to the RS_TrueExtremes function. (Note: The calling routine must declare the size of this array as 12-by-MaxNumDays, with MaxNumDays >= NumDays.)
Index	(Input) The number of the DataArray column in which the data for the current day is being loaded. This is an output of the RS_DailyDataArray function, and an input to the RS_TrueExtremes function.
oPrevTrHighest	(Output) The highest true-high of the previous NumDays (i.e., not including the current day.)
oPrevTrLowest	(Output) The lowest true-low of the previous NumDays (i.e., not including the current day.)
oPrevATR	(Output) The average true range for the previous NumDays (i.e., not including the current day.)

Remarks

This approach is intended primarily for use with RadarScreen indicators, where only one datastream can be used; in charting, similar calculations can be performed more easily using a multi-data approach.

Example

```
inputs: NumDays( 3 ) ;
variables: Index( 0 ), oPrevTrHighest( 0 ), oPrevTrLowest( 0 ), oPrevATR( 0 ) ;
arrays: DataArray[ 12, 100 ]( 0 ), SubArray[3]( 0 ) ;
Value1 = RS_DailyDataArray( NumDays, DataArray, Index, SubArray ) ;
if CurrentBar = 1 or Date <> Date[1] then
 Value2 = RS_TrueExtremes( NumDays, DataArray, Index, oPrevTrHighest,
  oPrevTrLowest, oPrevATR ) ;
Plot1( oPrevTrHighest ) ;
Plot2( oPrevTrLowest ) ;
```

RSI (Function)

Disclaimer

The RSI (Relative Strength Index) series function returns a value between zero and one hundred, regardless of the asset it is applied to. It is easy to calculate and keep track of. After calculating the initial RSI, only the previous day's data is required for the next calculation.

Syntax

```
RSI(Price, Length)
```

Returns

A numeric value containing RSI for the current bar.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to be considered.
Length	Numeric	Sets the number of bars to consider.

Remarks

The equation for the Relative Strength Index, RSI is:

RSI = 100 - (100/(1+RS))

RS = Average of 14 day's closes UP / Average of 14 day's closes DOWN

To calculate the first RSI value

1. Calculate the sum of the UP closes for the previous 14 days and divide this sum by 14. This is the average UP Close.
2. Calculate the sum of the DOWN closes for the previous 14 days and divide by 14. This is the average DOWN Close.
3. Divide the average UP Close by the average DOWN Close. This is the Relative Strength (RS).
4. Add 1.00 to the RS.
5. Divide the result obtained in Step 4 into 100.
6. Subtract the result obtained in Step 5 from 100. This is the first RSI.

To calculate the RSI each time following the first RSI value

1. To obtain the next average UP Close: Multiply the previous average Up Close by 13, add to this amount today's UP Close (if any) and divide the total by 14.
2. To obtain the next average DOWN: Multiply the previous average DOWN Close by 13, add to this amount today's DOWN Close (if any) and divide the total by 14.
3. Steps (3), (4), (5) and (6) are the same as for the initial RSI.

The input Price is usually hard coded with some bar attribute such as Close, Open, High, Low, and Volume or is replaced with a numeric series type input. However, it can be replaced with a valid EasyLanguage expression. For example: Close + Open, or Average(RSI(Close,14),14).

The input Length, just like Price, can be hard coded or replaced with a numeric simple type input.

Example

```
Plot1(RSI(Close,14));
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC

RSquare (Function)

Disclaimer

The RSquare function calculates the square of the Pearson product moment correlation coefficient, R.

Syntax

```
RSquare(Indep, Dep, Length)
```

Returns (Double)

A numeric value containing the square of the Pearson product moment correlation coefficient, R.

Parameters

Name	Type	Description
Indep	Numeric	Sets the independent value used in the calculation.
Dep	Numeric	Sets the dependent value used in the calculation.
Length	Numeric	Sets the number of bars to be considered in the calculation.

Remarks

R-Square (r^2) is the coefficient of determination, which measures the proportion of variation that is explained by the independent variable in a regression model. This function takes the value of CoeffR for the specified Length and squares the value. that is, CoeffR(Length) * CoeffR(Length). The RSquared value can be interpreted as the proportion of the variance of Price with respect to the specified period (‘Length‘).

Example

Assigns to Value1 the R-Square calculation based on 14 bars:

```
Value1 = RSquare(BarNumber, Close, 14);
```

Additional Example

If you wanted to identify a bar in which the R-Square had been rising for three consecutive bars, you could use the following syntax:

```
Value1 = RSquare(BarNumber, Close, 14);
If Value1 > Value1[1] AND Value1[1] > Value1[2] AND Value1[2] > Value1[3] Then
    Plot1(High, "R_risng");
```

RSquareArray (Function)

Disclaimer

The RSquareArray function calculates the square of the Pearson product moment correlation coefficient, R, between two arrays of values..

Syntax

```
RSquareArray(IndepArray, DepArray, Size)
```

Returns (Double)

A numeric value containing the square of the Pearson product moment correlation coefficient, R.

Parameters

Name	Type	Description
`IndDepArray`	Numeric	Specifies the name of a numeric array of the independent values.
`DepArray`	Numeric	Specifies the name of a numeric array of the dependent values.
`Size`	Numeric	Sets the number of data elements (size) in the array to be evaluated.

Usage

R-Square (r^2) is the coefficient of determination that measures the proportion of variation that is explained by the independent variables in a regression model. This function takes the value of CoeffR for the specified array, and squares that value. The RSquared value can be interpreted as a reflection of a linear relationship between two data sets.

The value for the `Size` input parameter should always be a whole number greater than 0 equal to the number of data elements in the array.

The `RSquareArray` function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Assigns to `Value1` the `RSquareArray` of one 10 element array to another 10 element array:

```
Array: myIndDepArray [10](0), myDepArray[10](0);
{… (assign values to both arrays) }
Value1 = RSquareArray (myIndDepArray, myDepArray, 10);
```

RunCommandOnLastBar (Function)

Disclaimer

The RunCommandOnLastBar function allows you to run a Command Line instruction or Macro from within any EasyLanguage study.

This function only runs the command line instructions or MACRO on the last bar of data.

EasyLanguage does not check the validity of the string that is specified for the keyword. If the command is not valid, a message will be displayed in the Events Log.

Syntax

```
RunCommandOnLastBar(CommandOrMacro);
```

Returns (Integer)

The value 1 when it runs on the last bar of data, otherwise it returns 0.

Parameters

Name	Type	Description
CommandOrMacro	String	Sets a string value that is either a command line MACRO, or an actual command line instruction sequence.

Remarks

You cannot historically back-test the results of orders place into the market with RunCommandOnLastBar.

Note: If the **Update value intra-bar** check box is enabled in an Indicator that calls the reserved word ‘RunCommand’, it could execute the command line instructions or MACRO on every tick, this is probably not what you are intending.

Examples

Run the ‘.CustBuyMarket’ MACRO if time is 10:00am and the Close price is greater than the Open price.

```
If Time = 1000 AND Close > Open Then
Value1=RunCommandOnLastBar(".CustBuyMarket MSFT, 500");
```

Run the function three times if Condition1 is TRUE. Each instruction set can contain one or more command sets.

```
If Condition1 Then Begin
Value1=RunCommandOnLastBar(".NWS");
Value1=RunCommandOnLastBar(".BuyLimit MSFT, 500 ;; .NewChart;; MSFT");
Value1=RunCommandOnLastBar(".BuyLimit DELL, 500 ;; .NewChart;; DELL");
End;
```

SessionCountDay (Function)

Disclaimer

The SessionCountDay function returns the number of available complete sessions for the specified day. Complete sessions both start and end during the same day, defined as 12 AM to 11:59 PM.

Syntax

```
SessionCountDay(SessionType,XDay);
```

Returns (Integer)

A numeric value for the current bar.

Parameters

Name	Type	Description
SessionType	Numeric	Sets the type of session to reference. 0 = Auto Detect, 1 = Regular Session
XDay	Numeric	Sets the day of the week that should be evaluated. 0=Sunday, 1=Monday, etc.

Remarks

The input parameter SessionType specifies the type of session information that should be returned. The type parameter may be specified as follows: Auto-Detect Session – [0, AutoSession] - uses whatever session is specified for the data series; regular or custom, or Regular Session – [1, RegularSession] - regular session information should always be returned.

Examples

Assigns to Value1 the number of available complete sessions for Wednesday:

```
Value1 = SessionCountDay(0,3);
```

See Also

PartSessionCountDay

SessionFirstBarTime (Function)

Disclaimer

The `SessionFirstbarTime` function returns the time of the first bar for the specified session.

Syntax

```
SessionFirstbarTime(SessionType,SessionNum);
```

Returns (Integer)

A numeric value for the current bar.

Parameters

Name	Type	Description
`SessionType`	Numeric	Sets the type of session to reference. 0 = Auto Detect, 1 = Regular Session
`SessionNum`	Numeric	Sets the session number to reference.

Remarks

The input parameter `SessionType` specifies the type of session information that should be returned. The type parameter may be specified as follows: Auto-Detect Session – [0, AutoSession] - uses whatever session is specified for the data series; regular or custom, or Regular Session – [1, RegularSession] - regular session information should always be returned. The input parameter `SessionNum` specifies the specific session number to be returned. This value ranges from 1 to the maximum number of sessions.

Examples

Assigns to `Value1` the time of the first bar for the specified session:

```
Value1 = SessionFirstbarTime (0,3);
```

ShowLongStop (Function)

Disclaimer

The ShowLongStop function adds text to a chart that displays the stop level on the chart for a long-side stop. This function is designed for use only with strategies.

Syntax

```
ShowLongStop(StopVal)
```

Returns (Integer)

In addition to displaying "Stop--", the function itself returns a numeric value representing the current market position (-1 for short, 1 for long, and 0 for no position).

Parameters

Name	Type	Description
StopVal	Numeric	Sets the stop value where the text will be displayed.

Remarks

The function adds the text "Stop--". If the stop value is beyond the upper/lower limit of the chart, the text will not appear unless the chart scaling is adjusted to include that area. In addition, the function returns the current market position that is assigned to a variable.

Example

Will display the "Stop--" at the lowest Low of the last two bars while the market position is long:

```
Value1 = ShowLongStop(Lowest(Low, 2));
```

Additional Example

If you wanted to show the stop text for a long trailing stop, you could use the following:

```
Variables: StopValue(0);

If Close Crosses Above Average(Close, 18) Then
 Buy This Bar on Close;

StopValue = Lowest(Low, 2);

If MarketPosition = 1 Then
 Sell Next Bar at StopValue Stop;

Value1 = ShowLongStop(StopValue);
```

ShowShortStop (Function)

Disclaimer

The ShowShortStop function adds text to a chart that displays the stop level on the chart for a short-side stop. This function is designed for use only with strategies.

Syntax

```
ShowShortStop(StopVal)
```

Returns (Integer)

In addition to displaying "Stop--", the function itself returns a numeric value representing the current market position (-1 for short, 1 for long, and 0 for no position).

Parameters

Name	Type	Description
StopVal	Numeric	Sets the stop value where the text will be displayed.

Remarks

The function adds the text "Stop--". If the stop value is beyond the upper/lower limit of the chart, the text will not appear unless the chart scaling is adjusted to include that area. In addition, the function returns the current market position that is assigned to a variable.

Example

Will display the "Stop--" at the lowest Low of the last two bars while the market position is long:

```
Value1 = ShowLongStop(Lowest(Low, 2));
```

Additional Example

If you wanted to show the stop text for a short trailing stop, you could use the following:

```
Variables: StopValue(0);

If Close Crosses Below Average(Close, 18) Then
 SellShort This Bar on Close;

StopValue = Highest(High, 2);

If MarketPosition = -1 Then
 BuyToCover Next Bar at StopValue Stop;

Value1 = ShowLongStop(StopValue);
```

Skew (Function)

Disclaimer

The Skew function calculates the skewness of a distribution for a set of values.

Syntax

```
Skew(Price, Length)
```

Returns (Double)

A numeric value containing the skewness of a distribution.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) on which the skew will be based.
Length	Numeric	Sets the number of bars to use to build the distribution.

Remarks

Skewness characterizes the degree of asymmetry of a distribution around its mean. Positive skewness indicates a distribution with an asymmetric tail extending toward more positive values. Negative skewness indicates a distribution with an asymmetric tail extending toward more negative values.

Example

Assigns to Value1 the skew of the distribution of the closes of the last 100 bars you can write:

```
Value1 = Skew(Close, 100);
```

SkewArray (Function)

Disclaimer

The SkewArray function calculates the skewness of a distribution for an array of values.

Syntax

```
SkewArray(PriceArray, Size)
```

Returns (Double)

A numeric value containing the skewness of a distribution for an array of values. If improper array sizing/reference occurs, the function will return a 0.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing values upon which the Skew is calculated.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.

Remarks

Skewness characterizes the degree of asymmetry of a distribution around its mean. Positive skewness indicates a distribution with an asymmetric tail extending toward more positive values. Negative skewness indicates a distribution with an asymmetric tail extending toward more negative values.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The SkewArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Note This array function is similar to Skew except that you can build an array of values that does not necessarily include contiguous bar values.

Example

Assigns to Value1 the Skew factor of the specified named array MyArray with 40 elements.

```
Array: MyArray[40](0);
```

{… add EL statements here to assign price values to array elements... }

```
Value1 = SkewArray(MyArray,40);
```

SkewOpt (Function)

Disclaimer

The SkewOpt function returns the optimizable skew for a set of values.

Syntax

```
SkewOpt(Price, Length, Mean, SDev)
```

Returns (Double)

A numeric value containing SkewOpt for the current bar. The function returns a zero if Length > 2 and Sdev < 0.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Length	Numeric	Sets the number of bars to consider in the calculation.
Mean	Numeric	Sets the average value for the calculation.
SDev	Numeric	Sets the standard deviation.

Example

```
Value1 = SkewOpt(24.50, 5, 22, 1);
```

SlowD (Function)

Disclaimer

The SlowD series function returns the slow D value for the Stochastic oscillator.

Syntax

```
SlowD(StochLength)
```

Returns (Double)

A numeric value containing SlowD for the current bar.

Parameters

Name	Type	Description
StochLength	Numeric	Sets the number of bars to consider.

Remarks

Please refer to the discussion under the Stochastic function.

Example

Assigns to Value1 the Stochastic SlowD over 14 bars.

```
Value1 = SlowD(14);
```

Reference

Takano, Mike. *Stochastic Oscillator, Technical Analysis of Stocks and Commodities*. April 1991.

Stein, John. *The Traders' Guide to Technical Indicators, Futures Magazine*. August 1990.

SlowDCustom (Function)

Disclaimer

The SlowDCustom series function returns the slow D value for the Stochastic oscillator based on user-defined price inputs.

Syntax

```
SlowDCustom(PriceH, PriceL, PriceC, StochLength)
```

Returns (Double)

A numeric value containing the SlowDCustom for the current bar.

Parameters

Name	Type	Description
PriceH	Numeric	Specifies which bar value (price, function, or formula) to use for the high in stochastic calculations.
PriceL	Numeric	Specifies which bar value (price, function, or formula) to use for the low in stochastic calculations.
PriceC	Numeric	Specifies which bar value (price, function, or formula) to use for the close in stochastic calculations.
StochLength	Numeric	Sets the number of bars to consider.

Remarks

Please refer to the discussion under the Stochastic function.

Example

Assigns to Value1 the Stochastic SlowD for offset high and low prices over 14 bars.

```
Value1 = SlowDCustom(High+1,Low-1,Close,14);
```

Reference

Takano, Mike. *Stochastic Oscillator, Technical Analysis of Stocks and Commodities*. April 1991.

Stein, John. *The Traders' Guide to Technical Indicators, Futures Magazine*. August 1990.

SlowDCustomOrig (Function)

Disclaimer

The SlowDCustomOrig series function returns the Slow D value for the Stochastic oscillator based on user-defined price inputs.

Syntax

```
SlowDCustomOrig(PriceH, PriceL, PriceC, StochLength, SmoothingLength1,
SmoothingLength2)
```

Returns (Double)

A numeric value containing the SlowDCustomOrig for the current bar.

Parameters

Name	Type	Description
PriceH	Numeric	Specifies which bar value (price, function, or formula) to use for the high in stochastic calculations.
PriceL	Numeric	Specifies which bar value (price, function, or formula) to use for the low in stochastic calculations.
PriceC	Numeric	Specifies which bar value (price, function, or formula) to use for the close in stochastic calculations.
StochLength	Numeric	Sets the number of bars to consider.
SmoothingLength1	Numeric	Sets the constant to use for smoothing the K line.
SmoothingLength2	Numeric	Sets the constant to use for smoothing the D line.

Remarks

This function differs from SlowDCustom by using the original smoothing method suggested by George Lane.

Please refer to the discussion under the Stochastic function.

Example

Assigns to Value1 the Stochastic SlowD for offset high and low prices over 14 bars.

```
Value1 = SlowDCustomOrig(High+1,Low-1,Close,14,3,3);
```

Reference

Takano, Mike. *Stochastic Oscillator, Technical Analysis of Stocks and Commodities*. April 1991.

Stein, John. *The Traders' Guide to Technical Indicators, Futures Magazine*. August 1990.

SlowK (Function)

Disclaimer

The SlowK series function returns the slow K value for the Stochastic oscillator.

Syntax

```
SlowK(StochLength)
```

Returns (Double)

A numeric value containing SlowK for the current bar.

Parameters

Name	Type	Description
StochLength	Numeric	Sets the number of bars to consider.

Remarks

Please refer to the discussion under the Stochastic function.

Example

Assigns to Value1 the Stochastic SlowK over 14 bars.

```
Value1 = SlowK(14);
```

Reference

Takano, Mike. *Stochastic Oscillator, Technical Analysis of Stocks and Commodities*. April 1991.

Stein, John. *The Traders' Guide to Technical Indicators, Futures Magazine*. August 1990.

SlowKCustom (Function)

Disclaimer

The SlowKCustom series function returns the Slow K value for the Stochastic oscillator based on user-defined price inputs.

Syntax

```
SlowKCustom(PriceH, PriceL, PriceC, StochLength)
```

Returns (Double)

A numeric value containing the SlowKCustom for the current bar.

Parameters

Name	Type	Description
PriceH	Numeric	Specifies which bar value (price, function, or formula) to use for the high in stochastic calculations.
PriceL	Numeric	Specifies which bar value (price, function, or formula) to use for the low in stochastic calculations.
PriceC	Numeric	Specifies which bar value (price, function, or formula) to use for the close in stochastic calculations.
StochLength	Numeric	Sets the number of bars to consider.

Remarks

Please refer to the discussion under the Stochastic function.

Example

Assigns to Value1 the Stochastic SlowK for offset high and low prices over 14 bars.

```
Value1 = SlowKCustom(High+1,Low-1,Close,14);
```

Reference

Takano, Mike. *Stochastic Oscillator, Technical Analysis of Stocks and Commodities*. April 1991.

Stein, John. *The Traders' Guide to Technical Indicators, Futures Magazine*. August 1990.

SlowKCustomOrig (Function)

Disclaimer

The SlowKCustomOrig series function returns the Slow K value for the Stochastic oscillator based on user-defined price inputs.

Syntax

```
SlowKCustomOrig(PriceH, PriceL, PriceC, StochLength, SmoothingLength)
```

Returns (Double)

A numeric value containing the SlowKCustomOrig for the current bar.

Parameters

Name	Type	Description
PriceH	Numeric	Specifies which bar value (price, function, or formula) to use for the high in stochastic calculations.
PriceL	Numeric	Specifies which bar value (price, function, or formula) to use for the low in stochastic calculations.
PriceC	Numeric	Specifies which bar value (price, function, or formula) to use for the close in stochastic calculations.
StochLength	Numeric	Sets the number of bars to consider.
SmoothingLength	Numeric	Sets the constant to use for smoothing the K line.

Remarks

This function differs from SlowKCustom by using the original smoothing method suggested by George Lane.

Please refer to the discussion under the Stochastic function.

Example

Assigns to Value1 the Stochastic FastK for offset high and low prices over 14 bars.

```
Value1 = SlowKCustomOrig(High+1,Low-1,Close,14,3);
```

Reference

Takano, Mike. *Stochastic Oscillator, Technical Analysis of Stocks and Commodities*. April 1991.

Stein, John. *The Traders' Guide to Technical Indicators, Futures Magazine*. August 1990.

SmoothedAverage (Function)

Disclaimer

The SmoothedAverage series function further smoothes an average of the last *x* bars. It does this by using the previous value of itself. This function is used in the same way as the Average function.

Syntax

```
SmoothedAverage(Price, Length)
```

Returns (Double)

A numeric value containing the Smoothed Average of values over a specified number of bars.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Length	Numeric	Sets the number of bars to consider.

Remarks

On the first bar, this function adds together all the values returned by the input Price for the specified Length, divides the sum by the Length, then stores the value in a variable called SUM. Only on the first bar is the SmoothedAverage function equal to the value stored in SUM.

SUM = Summation(Price, Length)

On each bar there after, the function uses a different equation.

```
SmoothedAverage = (Sum[1] - SmoothedAverage[1] + Price)/Length
```

Example

```
Plot1(SmoothedAverage(C,20);
```

Sort2DArray (Function)

Disclaimer

The Sort2DArray function performs a linked sort of values in a two dimensional array, using the primary dimension as the key.

Syntax

```
Sort2DArray(PriceArray, Size1, Size2, HiLo)
```

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing values upon which the sort is performed.
Size1	Numeric	Sets the number of array elements (size) in the 1st dimension to be sorted.
Size2	Numeric	Sets the number of array elements (size) in the 2nd dimension to include in the sort.
HiLo	Numeric	Sets the sort order. 1 = sort descending, -1 = sort ascending

Returns (Double)

The order of values of the two-dimensional array specified in PriceArray are changed as the result of running Sort2DArray. The Sort2DArray function itself returns 1.

Remarks

This function is used to change the order of values in an array by sorting them in either ascending or descending order based on the HiLo input. The sort is performed using the elements of the 1st dimension as a key.

The value for the Size1 and Size2 input parameters should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The Sort2DArray function only works with two-dimensional arrays. All array-based referencing begins with array element 1.

Example

Performs an ascending sort of the user declared two-dimensional array..

```
Array: myArray[30,4](0);
{… (assign values to array) }
Value1 = Sort2DArray(myArray, 30, 4, -1);
```

SortArray (Function)

Disclaimer

The SortArray function sorts the values of a one-dimensional array in either ascending or descending order.

Syntax

```
SortArray(PriceArray, Size, HiLo)
```

Returns (Double)

The order of values of the array specified in `PriceArray` are changed as the result of running `SortArray`. The `SortArray` function itself returns 1.

Parameters

Name	Type	Description
`PriceArray`	Numeric Array	Specifies the name of a numeric array containing values upon which the sort is performed.
`Size`	Numeric	Sets the number of data elements (size) in the array to evaluate.
`HiLo`	Numeric	Sets the sort order. 1 = sort descending, -1 = sort ascending

Remarks

This function is used to change the order of values in an array by sorting them in a specified order.

The value for the `Size` input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The `SortArray` function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Performs an ascending sort of the user declared array..

```
Array: myArray[30](0);
{… (assign values to array) }
Value1 = SortArray(myArray, 30,-1);
```

SortHeap2DArray (Function)

Disclaimer

The SortHeap2DArray function performs a linked heap sort of values in a two dimensional array, using the primary dimension as the key.

Syntax

```
SortHeap2DArray(PriceArray, Size1, Size2, Order)
```

Returns (Boolean)

The order of values of the two-dimensional array specified in PriceArray are changed as the result of running SortHeap2DArray. The SortHeap2DArray function itself returns True.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing values upon which the sort is performed.
Size1	Numeric	Sets the number of array elements (size) in the 1st dimension to be sorted.
Size2	Numeric	Sets the number of array elements (size) in the 2nd dimension to include in the sort.
Order	Numeric	Sets the sort order. 1 = sort descending, -1 = sort ascending

Remarks

This function is used to change the order of values in an array by sorting them in either ascending or descending order based on the HiLo input. The sort is performed using the elements of the 1st dimension as a key.

As a rule of thumb, consider using this heap sort, SortHeap2DArray, rather than Sort2DArray, when the array to be sorted contains more than 25 elements in its second dimension and speed is the primary concern. Be aware, however, that Sort2DArray and SortHeap2DArray may not return the same sorted array in all cases in which there are identical elements in the first dimension of the array. This is because both sorts are "unstable" sorts.

The value for the Size1 and Size2 input parameters should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The SortHeap2DArray function only works with two-dimensional arrays. All array-based referencing begins with array element 1.

Example

Performs an ascending sort of the user declared two-dimensional array..

```
Array: myArray[30,4](0);
```

{… (assign values to array) }

```
Value1 = SortHeap2DArray(myArray, 30, 4, -1);
```

SortHeapArray (Function)

Disclaimer

The SortHeapArray performs a heap sort on the values of a one-dimensional array in either ascending or descending order.

Syntax

```
SortHeapArray(PriceArray, Size, Order)
```

Returns (Boolean)

The order of values of the array specified in PriceArray are changed as the result of running SortHeapArray. The SortHeapArray function itself returns True.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing values upon which the sort is performed.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.
Order	Numeric	Sets the sort order. 1 = sort descending, -1 = sort ascending

Remarks

SortHeapArray may be better suited for sorting of single-dimensional arrays of more than 25 elements compared to the alternate function SortArray (that may be faster for arrays of 25 or fewer elements).

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The SortHeapArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Performs an ascending sort of the user declared array..

```
Array: myArray[30](0);
{… (assign values to array) }
Value1 = SortHeapArray(myArray, 30,-1);
```

StandardDev (Function)

Disclaimer

The `StandardDev` function calculates a standard deviation of values (population or sample).

Syntax

```
StandardDev(Price, Length, DataType)
```

Returns

A numeric value containing the current Standard Deviation.

Parameters

Name	Type	Description
`Price`	Numeric	Specifies which bar value (price, function, or formula) to be considered.
`Length`	Numeric	Sets the number of bars to be considered.
`DataType`	Numeric	1 = population, 2 = sample

Remarks

To derive the standard deviation, first find the variance and then take its square root:

$$\text{StdDev} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (D_i - M)^2}$$

…where

N is the number of elements

D is the individual elements in the sample

M is the sample mean

Example

Assigns to Value1 the sample type standard deviation `Close` over 21 bars.

```
Value1 = StandardDev(Close , 21, 2);
```

StandardDevAnnual (Function)

Disclaimer

The `StandardDevAnnual` function calculates a standard deviation of values, and presents an annualized number.

Syntax

```
StandardDevAnnual(Price, Length, DataType)
```

Returns (Double)

A numeric value containing the annualized Standard Deviation.

Parameters

Name	Type	Description
`Price`	Numeric	Specifies which bar value (price, function, or formula) to be considered.
`Length`	Numeric	Sets the number of bars to be considered.
`DataType`	Numeric	1 = population, 2 = sample

Remarks

The standard deviation is derived by first finding the variance and then taking its square root:

$$\text{StdDev} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(D_i-M)^2}$$

…where

N is the number of elements

D is the individual elements in the sample

M is the sample mean

This function then multiplies the standard deviation by `BarAnnualization` in order to generate an annualized value.

Example

Assigns to `Value1` the annualized standard deviation of the `Close` over a sample of 21 bars.

```
Value1 = StandardDevAnnual(Close, 21, 2);
```

StandardDevArray (Function)

Disclaimer

The StandardDevArray function calculates a standard deviation of array values.

Syntax

```
StandardDevArray(PriceArray, Size, DataType)
```

Returns (Double)

A numeric value containing the Standard Deviation for the specified array.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array upon which the standard deviation is calculated.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.
DataType	Numeric	Sets what data elements should be based on: 1 = population, 2 = sample

Remarks

To derive the standard deviation, first find the variance and then take its square root:

$$\text{StdDev} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(D_i - M)^2}$$

…where

N is the number of elements

D is the individual elements in the sample

M is the sample mean

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The SummationSqrArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

The following assigns the lowest value of the 20 elements in the array to Value1:

```
Array: myArray[20](0);
{… add EL statements here to assign values to array elements... }
Value1 = StandardDevArray(MyArray, 20);
```

See Also

ExtremesArray

Standardize (Function)

Disclaimer

The Standardize function returns a normalized value from a distribution characterized by the mean based on the `Price` for `Length` bars, and standard deviation.

Syntax

```
Standardize(Price, Length, NumDevs)
```

Parameters

Name	Type	Description
`Price`	Numeric	Specifies which bar value (price, function, or formula) to use.
`Length`	Numeric	Set the number of bars to be considered.
`NumDevs`	Numeric	Sets the number of standard deviations to consider in the standardization.

Returns (Double)

A numeric value containing the normalized value from a distribution characterized by the mean based on the `Price` for `Length` bars, and standard deviation (`NumDevs`).

Remarks

The normalized value is based on a distribution with a mean of 0 and a standard deviation of 1.

Example

Assigns to `Value1` the standardization of the `Close` over the last 14 bars, using 2 standard deviations.

```
  Value1 = Standardize(Close, 14, 2);
```

Assigns to `Value2` the standardization of the `High` over the last 10 bars, using 1.5 standard deviations:

```
  Value2 = Standardize(High, 10, 1.5);
```

StandardizeArray (Function)

Disclaimer

The StandardizeArray function returns a normalized value based on a specified number of standard deviations and a mean average.

Syntax

```
StandardizeArray(PriceArray, Size, NumDevs)
```

Returns (Double)

A numeric value containing a normalized value from an array, as characterized by the array-based mean and standard deviation

Parameters

Name	Type	Description
PriceArray	Numeric	Specifies the name of a numeric array containing values upon which the sum of squares of deviations is calculated.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.
NumDevs	Numeric	Sets the number of standard deviations to use.

Remarks

The mean and standard deviation used in the calculation of the standardized value are based on values in the specified price array.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The StandardizeArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Assigns to Value1 the Skew factor of the specified named array MyArray with 10 elements.

```
Array: MyArray[10](0);
{… add EL statements here to assign price values to array elements... }
Value1 = StandardizeArray(MyArray,10);
```

StdDev (Function)

Disclaimer

The `StdDev` function calculates a standard deviation of a population of values.

Syntax

```
StdDev(Price, Length)
```

Returns (Double)

A numeric value containing the current standard deviation.

Parameters

Name	Type	Description
`Price`	Numeric	Specifies which bar value (price, function, or formula) to be considered.
`Length`	Numeric	Sets the number of bars to be considered.

Remarks

To find the standard deviation, first find the variance and then take its square root:

$$\text{StdDev} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(D_i-M)^2}$$

…where

N is the number of elements

D is the individual elements in the sample

M is the sample mean

Example

Assigns to `Value1` the standard deviation of `Close` over 10 bars.

```
Value1 = StdDev(Close, 10);
```

StdDevS (Function)

Disclaimer

The StdDevS function calculates a standard deviation for a sample of values.

Syntax

```
StdDevS(Price, Length)
```

Returns (Double)

A numeric value containing the sample standard deviation value of the specified bar.

Parameters

Price	Specifies the price information to be considered.
Length	Indicates the length of time (in bars) to be considered.

Usage

This function returns a value representing how widely dispersed individual (Price) values are away from the mean average (using the same inputs) of a sample. To find the standard deviation, first find the variance of a sample and then take its square root.

Example

Assigns to Value1 the sample standard deviation of closing prices over the last 21 bars:

```
Value1 = StdDevS(Close, 21);
```

Assigns to Value2 the sample standard deviation of a 14 bar moving average of the High, calculated over the last 10 bars.

```
Value2 = StdDevS(Average(High, 14), 10);
```

StdError (Function)

Disclaimer

The StdError function returns a measure of the standardized variation around the regression line.

Syntax

```
StdError(Price, Length)
```

Returns (Double)

A numeric value containing the measure of the standardized variation around the regression line which is based on the Price and Length.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to be considered.
Length	Numeric	Sets the number of bars to be considered.

Remarks

This calculation provides an approximation of the accuracy of the regression line.

Example

Assigns to Value1 the standard error based on the Close over the past 14 bars:

```
Value1 = StdError(Close, 14);
```

Assigns to Value2 the standard error based on the High over the last 21 bars:

```
Value2 = StdError(High, 21);
```

StdErrorArray (Function)

Disclaimer

The StdErrorArray function calculates the standard error of values in an array.

Syntax

```
StdErrorArray(PriceArray, Size)
```

Returns (Double)

The standard error of values in an array..

Parameters

Name	Type	Description
`PriceArray`	Numeric Array	Specifies the name of a numeric array containing values upon which the standard error is calculated.
`Size`	Numeric	Sets the number of data elements (size) in the array to evaluate.

Remarks

The standard error calculation is a measure of the standardized variation around the regression line, as based upon the specified array. This calculation provides an approximation of the accuracy of the regression line.

The value for the `Size` input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The `StdErrorArray` function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Assigns to `Value1` the 10 bar moving average of the specified array.

```
Array: myArray[10](0);
{… (assign values to array) }
Value1 = StdErrorArray(myArray, 10);
```

StdErrorArray2 (Function)

Disclaimer

The StdErrorArray2 function calculates the standard error (the predicted y-value) between two arrays.

Syntax

```
StdErrorArray2(IndepArray, DepArray, ArraySz)
```

Returns (Double)

The standard error estimate of two user declared arrays.

Parameters

Name	Type	Description
IndepArray	Numeric Array	Specifies the name of a numeric array containing independent values upon which the standard error is calculated.
DepArray	Numeric Array	Specifies the name of a numeric array containing dependent values upon which the standard error is calculated.
ArraySz	Numeric	Sets the number of data elements (size) in the array to evaluate.

Remarks

The standard error calculation is a measure of the standardized variation around the regression line, as based upon the specified arrays. This calculation provides an approximation of the accuracy of the regression line.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The StdErrorArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

Assigns to Value1 the 20 bar standardized error of the specified arrays.

```
Array: myIndArray[20](0), myDepArray[20](0);
{… (assign values to array) }
Value1 = StdErrorArray2(myIndArray, myDepArray, 20);
```

Stochastic (Function)

Disclaimer

The Stochastic series function returns the four core values (FastK, FastD, SlowK and SlowD) associated with the Stochastic oscillator.

Syntax

```
Stochastic(PriceH, PriceL, PriceC, StochLength, Length1, Length2, SmoothingType,
oFastK, oFastD, oSlowK, oSlowD)
```

Returns (Integer)

The oFastK, oFastD, oSlowK, and oSlowD output parameters return the fast and slow averages for the K and D lines. The Stochastic function itself returns 1 if successful, otherwise -1 if both FastK=0 and SlowK=0.

Parameters

Parameters

Name	Type	Description
PriceH	Numeric	Specifies which bar value (price, function, or formula) to use for the high in stochastic calculations.
PriceL	Numeric	Specifies which bar value (price, function, or formula) to use for the low in stochastic calculations.
PriceC	Numeric	Specifies which bar value (price, function, or formula) to use for the close in stochastic calculations.
StochLength	Numeric	Sets the number of bars to consider.
Length1	Numeric	Sets the constant for smoothing the fast K line.
Length2	Numeric	Sets the constant for smoothing the fast D line.
SmoothingType	Numeric	Sets the calculation method for smoothing: 1 for calculations based on the original formula. 2 to conform to legacy TradeStation calculations..
oFastK	Numeric	Outputs the value for the fast K line.
oFastD	Numeric	Outputs the value for the fast D line.
oSlowK	Numeric	Outputs the value for the slow K line.
oSlowD	Numeric	Outputs the value for the slow D line.

Remarks

The Stochastic oscillators indicate overbought and oversold areas in the market, based upon momentum or price velocity.

Stochastics are made up of four individual calculations: FastK, FastD, SlowK, SlowD. The core of the calculations is based upon the FastK value that is calculated in the following manner.

`FastK = (C - L)/(H - L)*100`... where

L = Lowest Low of a specified period

H = Highest High of a specified period

C = Current bar Close

FastD is a smoothed average of FastK, SlowK is equal to FastD, and SlowD is a further smoothed SlowK.

See Multiple Output Function for more information on using output parameters to return values.

Example

Outputs the FastK, FastD, SlowK, and SlowD stochastic values to the declared variables over 14 bars.

```
Vars: oFastK(0), oFastD(0), oSlowK(0), oSlowD(0);

Value1 = Stochastic(H, L, C, 14, 3, 3, 1, oFastK, oFastD, oSlowK, oSlowD);
```

Reference

Takano, Mike. *Stochastic Oscillator, Technical Analysis of Stocks and Commodities*. April 1991.

Stein, John. *The Traders' Guide to Technical Indicators, Futures Magazine*. August 1990.

StochasticExp (Function)

Disclaimer

The StochasticExp series function returns the exponentially smoothed values FastD and SlowD.

Syntax

```
StochasticExp(PriceH, PriceL, PriceC, StochLength, SmoothingLength1,
SmoothingLength2, oFastK, oSlowD)
```

Parameters

Parameters

Name	Type	Description
PriceH	Numeric	Specifies which bar value (price, function, or formula) to use for the high in stochastic calculations.
PriceL	Numeric	Specifies which bar value (price, function, or formula) to use for the low in stochastic calculations.
PriceC	Numeric	Specifies which bar value (price, function, or formula) to use for the close in stochastic calculations.
StochLength	Numeric	Sets the number of bars to consider.
SmoothingLength1	Numeric	Sets the constant for smoothing the fast K line.
SmoothingLength2	Numeric	Sets the constant for smoothing the fast D line.
oFastD	Numeric	Outputs the value for the fast D line.
oSlowD	Numeric	Outputs the value for the slow D line.

Returns (Integer)

The oFastD and oSlowD output parameters return the fast and slow averages for the D line. The StochasticExp function itself returns 1.

Usage

The Stochastic oscillators indicate overbought and oversold areas in the market, based upon momentum or price velocity.

Stochastics are made up of four individual calculations: FastK, FastD, SlowK, SlowD. The core of the calculations is based upon the FastK value that is calculated in the following manner.

FastK = (C - L)/(H - L)*100… where

L = Lowest Low of a specified period

H = Highest High of a specified period

C = Current bar Close

FastD is a smoothed average of FastK, SlowK is a value equal to FastD, and SlowD is a further smoothed SlowK.

See Multiple Output Function for more information on using output parameters to return values.

Example

Outputs the FastD and SlowD stochastic values to the declared variables over 14 bars using smoothing values of 3..

```
Vars: oFastD(0), oSlowD(0);
Value1 = StochasticExp(H, L, C, 14, 3, 3, oFastD, oSlowD);
```

Reference

Takano, Mike. *Stochastic Oscillator, Technical Analysis of Stocks and Commodities*. April 1991.

Stein, John. *The Traders' Guide to Technical Indicators, Futures Magazine*. August 1990.

StopLimitOrder (Function)

Disclaimer

The StopLimitOrder function is used to configure and send a limit if touched order using the order entry macro **.PlaceOrder**. You can call this function directly from your own EasyLanguage code to simplify the formatting and generation of macro orders.

NOTE Care should be exercised when calling this function as it is intended to send live orders. Confirmations for macro-generated orders can be configured by using the File -> Preferences -> TradeStation Order Entry menu sequence.

Syntax

```
StopLimitOrder(Frequency,Account,Action,SymbolCategory,Symbol,Quantity,Duration,GTDD
ate,StopPrice,LimitPrice)
```

Returns (Integer)

StopLimitOrder returns 1 if called on a "real-time" tick and if a trade is allowed based on the user input "Frequency". The function returns -1 (negative one) in other cases; for example, if the function is called on an historical bar. If there are errors in the order parameters a runtime error message will be generated and the order will not be placed.

Parameters

Name	Type	Description	Supported Values
Frequency	String	Sets the order frequency.	Once, OncePerBar, EndOfBar, Always
Account	String	Sets the TradeStation account to be used for this order.	*Account string*
Action	String	Sets the order action.	Buy, Sell, SellShort, BuyToCover, BuyToOpen, BuyToClose, SellToOpen, SellToClose
SymbolCategory	String	Sets the trading category for the symbol'.	Equity, Future, Forex, EquityOption, FutureOption
Symbol	String	Sets the symbol to be used for this order.	*Symbol string*
Quantity	Numeric	Sets the number of shares, contracts, lots, etc. to be placed for this order.	Number
Duration	String	Sets the order duration.	Day, Day+, GTC, GTC+, GTD, GTD+, IOC, FOK, OPG, 1Min, 3Min, 5Min
GTDDate	String	Sets the date, if appropriate, to be used with the specified duration.	*Date Format* (MM/DD/YY).
StopPrice	Numeric	Sets the Stop price to be used for this order.	Number
LimitPrice	Numeric	Sets the Limit price to be used for this order.	Number

Remarks

The StopLimitOrder function uses the PlaceOrder function to format the order parameters and call the order entry macro. The PlaceOrder function is only intended to be used by this and other designated TradeStation functions.

This function disables advanced order placement features (All or None, Buy on minus,Sell on plus, etc.).

See .PlaceOrder command for more information on using the order entry macro for placing orders.

Example

Places a sell limit order of 100 shares for MSFT at a limit price of 24.35 using the order entry macro .PlaceOrder. Value1 returns a 1 if the order is valid.

```
Value1 =
StopLimitOrder("Once","SIM15180","Sell","Equity","MSFT",100,"Day","",24.65,24.35);
```

See Also

LimitIfTouchedOrder, LimitIOrder, MarketIfTouchOrder, MarketOrder, StopMarketOrder, TrailingStopOrder

StopMarketOrder (Function)

Disclaimer

The `StopMarketOrder` function is used to configure and send a limit if touched order using the order entry macro **.PlaceOrder**. You can call this function directly from your own EasyLanguage code to simplify the formatting and generation of macro orders.

> **NOTE** Care should be exercised when calling this function as it is intended to send live orders. Confirmations for macro-generated orders can be configured by using the File -> Preferences -> TradeStation Order Entry menu sequence.

Syntax

```
StopMarketOrder(Frequency,Account,Action,SymbolCategory,Symbol,Quantity,Duration,GTD
Date,StopPrice)
```

Returns (Integer)

`StopMarketOrder` returns 1 if called on a "real-time" tick and if a trade is allowed based on the user input "Frequency". The function returns -1 (negative one) in other cases; for example, if the function is called on an historical bar. If there are errors in the order parameters a runtime error message will be generated and the order will not be placed.

Parameters

Name	Type	Description	Supported Values
`Frequency`	String	Sets the order frequency.	Once, OncePerBar, EndOfBar, Always
`Account`	String	Sets the TradeStation account to be used for this order.	*Account string*
`Action`	String	Sets the order action.	Buy, Sell, SellShort, BuyToCover, BuyToOpen, BuyToClose, SellToOpen, SellToClose
`SymbolCategory`	String	Sets the trading category for the symbol'.	Equity, Future, Forex, EquityOption, FutureOption
`Symbol`	String	Sets the symbol to be used for this order.	*Symbol string*
`Quantity`	Numeric	Sets the number of shares, contracts, lots, etc. to be placed for this order.	Number
`Duration`	String	Sets the order duration.	Day, Day+, GTC, GTC+, GTD, GTD+, IOC, FOK, OPG, 1Min, 3Min, 5Min
`GTDDate`	String	Sets the date, if appropriate, to be used with the specified duration.	*Date Format* (MM/DD/YY).
`StopPrice`	Numeric	Sets the Stop price to be used for this order.	Number

Remarks

The `StopMarketOrder` function uses the `PlaceOrder` function to format the order parameters and call the order entry macro. The `PlaceOrder` function is only intended to be used by this and other designated TradeStation functions.

This function disables advanced order placement features (All or None, Buy on minus,Sell on plus, etc.).

See .PlaceOrder command for more information on using the order entry macro for placing orders.

Example

Places a sell limit order of 100 shares for MSFT at a limit price of 24.35 using the order entry macro .PlaceOrder. Value1 returns a 1 if the order is valid.

```
Value1 =
StopMarketOrder("Once","SIM15180","Sell","Equity","MSFT",100,"Day","",24.65);
```

See Also

LimitIfTouchedOrder, LimitIOrder, MarketIfTouchOrder, MarketOrder, StopLimitOrder, TrailingStopOrder

StrColorToNum (Function)

Disclaimer

The `StrColorToNum` function returns the color number for a specific color name.

Syntax

```
StrColorToNum(Color)
```

Returns

A numeric value containing the color number of a color name.

Parameters

Name	Type	Description
`Color`	Numeric	Sets a text string containing a color name.

Remarks

A numeric value containing the color number of a color name.

Example

Assigns to `Value1` the color number 6 for the color name "Red".

```
Value1 = StrColorToNum("Red");
```

Summation (Function)

Disclaimer

The Summation function adds a series of numbers together over a specified number of bars.

Syntax

```
Summation(Price, Length)
```

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to sum.
Length	Numeric	Sets the number of bars to consider.

Returns (Double)

A numeric value containing the sum of the last Length occurrences of Price.

Remarks

The number of values in the series is determined by the input Length. The values added together are determined by the input Price. For example, if the inputs Price and Length were replaced by Close and 14, respectively, the function would add together the last fourteen closes.

Note The current bar's Price is included in the calculation, unless the function is offset.

Example

```
Plot1(Average(Close,9));
```

See Also

SummationFC

SummationArray (Function)

Disclaimer

This SummationArray function performs a summation of an array of values.

Syntax

```
SummationArray(PriceArray, Size)
```

Returns (Double)

A numeric value containing the sum of an array of values. If improper array sizing/reference occurs, the function will return a -1.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array upon which the summation is calculated.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.

Remarks

This function accumulates the values of the array and returns the sum total.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The SummationArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

The following assigns the lowest value of the 20 elements in the array to Value1:

```
Array: myArray[20](0);
{… add EL statements here to assign values to array elements... }
Value1 = SummationArray(MyArray, 20);
```

See Also

ExtremesArray, DevSqrdArray

SummationFC (Function)

Disclaimer

The SummationFC series function uses a fast calculation method to add a series of prices over a specified number of bars.

Syntax

```
SummationFC(Price, Length)
```

Returns (Double)

A numeric value containing the sum of the last Length occurrences of Price.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to sum.
Length	Numeric	Sets the number of bars to use in the calculation.

Remarks

The number of values in the series is determined by the input Length. The values added together are determined by the input Price. For example, if the inputs Price and Length were replaced by Close and 14, respectively, the function would add together the last fourteen closes.

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as: (High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

Note This series function returns exactly the same values as Summation except that it uses a fast calculation method that takes slightly more memory than the non-FC version.

Example

```
Plot1(AverageFC(Close,9));
```

See Also

Summation

SummationIf (Function)

Disclaimer

The SummationIf function performs a summation of price when a condition is true.

Syntax

```
SummationIf(Test, Price, Length)
```

Returns (Double)

A numeric value containing the sum of the last Length occurrences of Price if Test is True.

Parameters

Name	Type	Description
Test	Numeric	Specifies the criteria that must be true for summation to occur.
Price	Numeric	Specifies which bar value (price, function, or formula) to sum.
Length	Numeric	Sets the number of bars used in the calculation.

Remarks

The number of values in the series is determined by the input Length. The values added together are determined by the input Price. For example, if the inputs Price and Length were replaced by Close and 14, respectively, the function would add together the last fourteen closes.

Note The current bar's Price is included in the calculation, unless the function is offset.

Example

```
Value1 = SummationIf(High>Close[1], High-Close, 14);
```

SummationRecArray (Function)

Disclaimer

The SummationRecArray function performs a summation of the reciprocal value of array elements.

Syntax

```
SummationRecArray(PriceArray, Size)
```

Returns (Double)

A numeric value containing the sum of the reciprocal values of array elements.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array upon which the reciprocal summation is calculated.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.

Remarks

This function accumulates the reciprocal values of the array and returns the sum total.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The SummationRecArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

The following assigns the lowest value of the 20 elements in the array to Value1:

```
Array: myArray[20](0);
{… add EL statements here to assign values to array elements... }
Value1 = SummationRecArray(MyArray, 20);
```

See Also

ExtremesArray

SummationSqrArray (Function)

Disclaimer

The SummationSqrArray function performs a summation of the square of array elements.

Syntax

```
SummationSqrArray(PriceArray, Size)
```

Returns (Double)

A numeric value containing the sum of the square of array elements.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array upon which the summation of squares is calculated.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.

Remarks

This function accumulates the squared values of the array and returns the sum total.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The SummationSqrArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

The following assigns the lowest value of the 20 elements in the array to Value1:

```
Array: myArray[20](0);
{… add EL statements here to assign values to array elements... }
Value1 = SummationSqrArray(MyArray, 20);
```

See Also

ExtremesArray

SwingHigh (Function)

Disclaimer

The SwingHigh function returns the high pivot price where a Swing High occurred.

Syntax

```
SwingHigh(Instance, Price, Strength, Length)
```

Returns (Integer)

A numeric value containing the high pivot price where the specified Swing High occurred, or -1 if not found.

Parameters

Name	Type	Description
Instance	Numeric	Sets which occurrence (that is, 1 = most recent, 2 = 2nd most recent, and so on) to return.
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Strength	Numeric	Sets the required number of bars on either side of the swing bar.
Length	Numeric	Sets the number of bars to be considered.

Remarks

A Swing High occurs when the Price of a bar is at least as high as the same Price on the preceding bar(s), and higher than the same Price on the bar(s) that follow it.

The input Strength is the number of bars on each side of the SwingHigh. A strength of one indicates that the value returned by the input Price must be greater than or equal to the same value returned for the bar on its left and greater than the bar on its right.

The input Length refers to the number of bars being examined for the SwingHigh.

The input Instance refers to which SwingHigh you want to use. For example, if in a twenty-one bar period three swing highs were found, it becomes necessary to specify which SwingHigh is desired. If the most recent SwingHigh is desired, a one (1) would be substituted for the input Instance.

Note If no SwingHigh is found in the period (Length) specified, the function will return a minus one (-1). The value of the input Length must exceed Strength by at least one. In addition, the Maximum number of bars referenced by a study (known as MaxBarsBack) must be greater than the sum of the values of Strength and Length.

Example

Assigns to Value1 the most recently occurring High in over the last 10 bars that has a strength of 4 on both the left and right sides of the swing bar..

```
Value1 = SwingHigh(1,Close,4,10);
```

SwingHighBar (Function)

Disclaimer

The SwingHighBar function returns the number of bars ago a Swing High bar occurred.

Syntax

```
SwingHighBar(Instance, Price, Strength, Length)
```

Returns (Integer)

A numeric value containing the number of bars ago the specified Swing High occurred, or -1 if not found.

Parameters

Name	**Type**	**Description**
Instance	Numeric	Sets which occurrence (that is, 1 = most recent, 2 = 2nd most recent, and so on) to return.
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Strength	Numeric	Sets the required number of bars on either side of the swing bar.
Length	Numeric	Sets the number of trailing bars to consider.

Remarks

A Swing High occurs when the Price of a bar is at least as high as the same Price on the preceding bar(s), and higher than the same Price on the bar(s) that follow it.

The input Strength is the number of bars on each side of the SwingHighBar. A strength of one indicates that the value returned by the input Price must be greater than or equal to the same value returned for the bar on its left and greater than the bar on its right.

The input Length refers to the number of bars being examined for the SwingHighBar.

The input Instance refers to which SwingHighBar you want to use. For example, if in a twenty-one bar period three swing highs were found, it becomes necessary to specify which SwingHighBar is desired. If the most recent SwingHighBar is desired, a one (1) would be substituted for the input Instance.

Note If no SwingHighBar is found in the period (Length) specified, the function will return a minus one (-1). The value of the input Length must exceed Strength by at least one. In addition, the Maximum number of bars referenced by a study (known as MaxBarsBack) must be greater than the sum of the values of Strength and Length.

Example

Plots the number of bars ago that the most recent swing bar high occurred based on the Close with 3 bars on either side of the swing within 10 trailing bars.

```
Plot1(SwingHighBar(1,Close,3,10);
```

SwingIndex (Function)

Disclaimer

The SwingIndex function returns a positive or negative value that represents the comparative up or down swing strength between two bar periods.

Function

SwingIndex

Returns (Double)

A numeric value containing the current Swing Index.

Parameters

None

Remarks

The SwingIndex function examines the following four cross-bar comparisons and one intra-bar comparison that are strong indicators of an up day or a down day:

1. Close today above or below previous Close.
2. Close today above or below Open today.
3. High today above or below previous Close.
4. Low today above or below previous Close.
5. Previous Close minus previous Open.

If the factors pointed toward an UP day, the SwingIndex would be a positive value from zero and one hundred. If the factors pointed toward a DOWN day, the function would return a value between zero and negative one hundred.

$$\text{Swing Index} = 50\left[\frac{C_2 - C_1 + .5(C_2 - O_2) + .25(C_1 - O_1)}{R}\right]\frac{K}{L}$$

…where

K = the larger of

- H_2-C_1
- L_2-C_1

L = value of a limit move in one direction.

To obtain R, first determine the largest of:

- H2-C1 – If the largest, $R=(H_2-C_1)-.5(L_2-C_1)+.25(C_{3-}O_1)$
- L2-C1 – If the largest, $R=(L_2-C_1)-.5(H_2-C_1)+.25(C_{3-}O_1)$
- H2-L2 – If the largest, $R=(H_2-L_2)+.25(C_{3-}O_1)$

Note Since only futures have a relative daily limit value, this function only makes sense if it is applied to a futures contract. If you use the Swing Index Indicator or Accumulation Swing Index Indicator and it only plots a zero flat line check the Daily Limit value.

Example

```
Plot1(SwingIndex);
```

Reference

Wilder, Welles, Jr. *New Concepts in Technical Trading Systems*. Trend Research. McLeansville, NC

SwingLow (Function)

Disclaimer

The SwingLow function returns the low pivot price where a Swing Low occurred.

Syntax

```
SwingLow(Instance, Price, Strength, Length)
```

Returns (Integer)

A numeric value containing the low pivot price where the specified Swing Low occurred, or -1 if not found.

Parameters

Name	**Type**	**Description**
Instance	Numeric	Sets which occurrence (that is, 1 = most recent, 2 = 2nd most recent, and so on) to return.
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Strength	Numeric	Sets the required number of bars on either side of the swing bar.
Length	Numeric	Sets the number of bars to be considered.

Remarks

A Swing Low occurs when the Price of a bar is at least as Low as the same Price on the preceding bar(s), and Lower than the same Price on the bar(s) that follow it.

The input Strength is the number of bars on each side of the SwingLow. A strength of one indicates that the value returned by the input Price must be less than or equal to the same value returned for the bar on its left and less than the bar on its right.

The input Length refers to the number of bars being examined for the SwingLow.

The input Instance refers to which SwingLow you want to use. For example, if in a twenty-one bar period three swing Lows were found, it becomes necessary to specify which SwingLow is desired. If the most recent SwingLow is desired, a one (1) would be substituted for the input Instance.

Note If no SwingLow is found in the period (Length) specified, the function will return a minus one (-1). The value of the input Length must exceed Strength by at least one. In addition, the Maximum number of bars referenced by a study (known as MaxBarsBack) must be greater than the sum of the values of Strength and Length.

Example

Assigns to Value1 the most recently occurring Close in over the last 10 bars that has a strength of 4 on both the left and right sides of the swing bar..

```
Value1 = SwingLow(1,Close,4,10);
```

SwingLowBar (Function)

Disclaimer

The SwingLowBar function returns the number of bars ago a Swing Low bar occurred.

Syntax

```
SwingLowBar(Instance, Price, Strength, Length)
```

Returns (Integer)

A numeric value containing the number of bars ago the specified Swing Low occurred, or -1 if not found.

Parameters

Name	Type	Description
Instance	Numeric	Sets which occurrence (that is, 1 = most recent, 2 = 2nd most recent, and so on) to return.
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Strength	Numeric	Sets the required number of bars on either side of the swing bar.
Length	Numeric	Sets the number of trailing bars to consider.

Remarks

A Swing Low occurs when the Price of a bar is at least as Low as the same Price on the preceding bar(s), and Lower than the same Price on the bar(s) that follow it.

The input Strength is the number of bars on each side of the SwingLowBar. A strength of one indicates that the value returned by the input Price must be less than or equal to the same value returned for the bar on its left and less than the bar on its right.

The input Length refers to the number of bars being examined for the SwingLowBar.

The input Instance refers to which SwingLowBar you want to use. For example, if in a twenty-one bar period three swing Lows were found, it becomes necessary to specify which SwingLowBar is desired. If the most recent SwingLowBar is desired, a one (1) would be substituted for the input Instance.

Note If no SwingLowBar is found in the period (Length) specified, the function will return a minus one (-1). The value of the input Length must exceed Strength by at least one. In addition, the Maximum number of bars referenced by a study (known as MaxBarsBack) must be greater than the sum of the values of Strength and Length.

Example

Plots the number of bars ago that the most recent swing bar low occurred based on the Close with 3 bars on either side of the swing within 10 trailing bars.

```
Plot1(SwingLowBar(1,Close,3,10);
```

Text_Exist (Function)

Disclaimer

The Text_Exist function identifies whether a specified text object exists in a chart.

Syntax

```
Text_Exist(TextId)
```

Returns (Integer)

The function returns True is the specified text object is found, otherwise False.

Parameters

Name	Type	Description
TextID	Numeric	Sets the ID number of a text object to find on a chart.

Example

Assigns to Condition1 a value of True or False based on whether the text object 2 is found.

```
Value1 = Text_Exist(2);
```

Text_Float (Function)

Disclaimer

The Text_Float series function maintains the location of a text object at fixed number of bars back from the last bar on the chart and a fixed percentage between the charts lowest and highest prices.

Syntax

```
Text_Float(TextID, TextBarsBack, TextPricePercent)
```

Returns (Integer)

This functions re-locates a text object based on the specified input parameters. The function itself returns a value of 1.

Parameters

Name	Type	Description
TextID	Numeric	Sets the ID number of an existing text object on a chart.
TextBarsBack	Numeric	Numeric value representing the number of bars back from right of chart at which to maintain location of text object
TextPricePercent	Numeric	Numeric value of a percentage used to determine the vertical location of the text object between the highest and lowest visible prices.

Remarks

The Text_Float function is used to float the position of a specified text object when new bars form, bar spacing is changed, or the range of prices displayed on the chart is changed.

The value of TextPricePercent should an integer from 0 to 100. The value of TextBarsBack should be an integer ranging from 0 (for the current bar) to the maximum number of bars back on the chart.

Example

This example creates a new text object containing the text "Target" and re-positions it to a floating location at the 5th bar from the right and 75% of the visible price range.

```
if LastBarOnChart then begin
  Value1 = Text_New(0, 0, 0, "Target");
  Value2 = Text_Float(Value1, 5, 75);
end;
```

TimeSeriesForecast (Function)

Disclaimer

The `TimeSeriesForecast` function returns the value where a linear regression line of the Close is projected to be relative to a future (or past) bar position.

Syntax

```
TimeSeriesForecast(Length, TgtBar)
```

Returns (Integer)

A numeric value of a future (or past) bar position.

Parameters

Name	Type	Description
`Length`	Numeric	Sets the bars to consider.
`TgtBar`	Numeric	Sets the point in the past or future to use for projecting a regression value. Use a positive integer for a previous point and a negative integer for a future point.

Remarks

Linear Regression is a concept also known as the "least squares method" or "best fit." Linear Regression attempts to fit a straight line between several data points in such a way that distance between each data point and the line is minimized.

The equation of any line resembles the following:

$$y = mx + b$$

In the equation *m* refers to the slope of the regression line, *b* refers to the constant intercept of the y-axis, *x* is the independent variable, and *y* is the dependent variable. This function returns all values.

Example

Assigns to `Value1` the projected value 5 bars in the future of a linear regression line over the past 20 bars.

```
Value1 = TimeSeriesForecast (20, -5);
```

Reference

Linear Regression is a principle found in most statistical publications.

TimeToMinutes (Function)

Disclaimer

The TimeToMinutes function calculates the number of minutes from midnight for any 24-hour (HHMM) time value.

Syntax

```
TimeToMinutes(XTime)
```

Returns (Integer)

A numeric value containing the number of minutes between midnight and XTime.

Parameters

Name	Type	Description
XTime	Numeric	Specifies a time in 24-hour format.

Remarks

This function allows for the proper addition and subtraction of time. The first step when adding or subtracting a number of minutes from time is to convert the time to minutes through the use of the TimeToMinutes function.

Then, add or subtract the number of minutes, and, finally convert the number of minutes back to time using the MinutesToTime function.

XTime is a numeric simple type input and can be hard coded with a number or can be replaced by a numeric simple input.

Example

Assigns to Value1 the number of minutes since midnight, 150 in this case, based on a 24-hour time input equivalent to 2:30am.

```
Value1 = TimeToMintues(0230);
```

TL_Exist (Function)

Disclaimer

The TL_Exist function returns true if a trendline exists.

Syntax

```
TL_Exist(TrendLineID)
```

Returns (Boolean)

True if the specified trendline (using TrendLineID) exists in the current chart. False if not.

Parameters

Name	Type	Description
TrendLineID	Numeric	Sets the identification number of the trendline.

Example

If you want to place a buy order to buy at the trendline number 1, only if the trendline exits, you can write:

```
If TL_Exist(1) Then
 Buy Next Bar at TL_GetValue(1, Date, Time) Stop;
```

TLAngle (Function)

Disclaimer

The TLAngle function returns the angle of a trendline.

Syntax

```
TLAngle(StartPrice, StartBar, EndPrice, EndBar)
```

Parameters

Name	Type	Description
StartPrice	Numeric	Sets the price of the trendline start point.
StartBar	Numeric	Sets the bar number of the trendline start point.
EndPrice	Numeric	Sets the price of the trendline end point.
EndBar	Numeric	Sets the bar number of the trendline end point.

Returns (Double0

A numeric value representing the angle of the specified trendline.

Remarks

To calculate the angle, the function requires that you specify the start and end bars for the trendline as well as the start and end prices. The inputs StartPrice and EndPrice are the start price and end price, respectively, of the trendline. They are usually replaced by values such as Close, High, Low, and so on, or are replaced with numeric series type inputs.

Also, you must offset the Price by its corresponding bar number. For example, if you want the trendline angle for a line drawn from the Close of ten bars ago to the Close of the current bar, replace the inputs StartPrice, StartBar, EndPrice, and EndBar with the values Close[10], 10, Close[0], and 0 respectively.

The inputs StartBar and EndBar refer to the bar numbers of the starting and ending points of the trendline. These inputs must be replaced by positive whole numbers or be replaced with numeric simple type inputs.

The formula used is a simple rise over run calculation to obtain the slope; the formula then takes the arctangent of that slope. In fact, you can obtain the same value as the TLAngle function by taking the ArcTangent of the TLSlope function.

Example

```
Plot1(TLAngle(High,1,High 9);
```

See Also

TLAngleEasy

TLAngleEasy (Function)

Disclaimer

The TLAngleEasy function returns the angle of a trendline with one price input.

Syntax

```
TLAngleEasy(Price, StartBar, EndBar)
```

Returns (Double)

A numeric value containing the angle of a trendline.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
StartBar	Numeric	Sets the bar number of the trendline start point.
EndBar	Numeric	Sets the bar number of the trendline end point.

Remarks

This function is similar to TLAngle, except that you cannot specify different prices for the start and end points. This function uses the same price on the start and end bars.

The formula used is a simple rise over run calculation to obtain the slope; the formula then takes the cotangent of that slope. In fact, you can obtain the same value as the TLAngleEasy function by taking the ArcTangent of the TLSlope function.

Example

```
Plot1(TLAngleEasy(High,1,9);
```

See Also

TLAngle

TLSlope (Function)

Disclaimer

The TLSlope function returns the slope of a trendline.

Syntax

```
TLSlope(StartPrice, StartBar, EndPrice, EndBar)
```

Returns (Double)

A numeric value containing the slope of the specified trendline.

Parameters

Name	Type	Description
StartPrice	Numeric	Sets the price of the trendline start point.
StartBar	Numeric	Sets the bar number of the trendline start point.
EndPrice	Numeric	Sets the price of the trendline end point.
EndBar	Numeric	Sets the bar number of the trendline end point.

Remarks

To calculate the angle the function will need the bar numbers and prices for the two endpoints of the trendline. The inputs EndPrice and StartPrice are the starting point and ending points of the trendline, respectively. They are usually replaced by values such as Close, High, Low, and so on, or can be replaced with numeric series inputs.

Note You must offset the price by its corresponding bar number. For example, if you wanted the trendline angle for a line drawn from the Close of ten bars ago to the Close of the current bar, replace the inputs StartPrice, StartBar, EndPrice, and EndBar would be replaced with the values Close[10], 10, Close[0], and 0 respectively.

The inputs EndBar and StartBar refer to the bar numbers of the starting and ending points of the trendline. These inputs can be replaced by whole numbers or with numeric simple type inputs that return positive whole numbers when using their default values. The formula used is a simple rise over run calculation.

Example

```
Plot1(TLSlope(High,1,High,9));
```

See Also

TLSlopeEasy

TLSlopeEasy (Function)

Disclaimer

The TLSlopeEasy function returns the slope of a trendline, if that trendline were to be drawn between two data points on your chart.

Syntax

```
TLSlopeEasy(Price, StartBar, EndBar)
```

Returns (Double)

A numeric value containing the slope of a trendline.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
StartBar	Numeric	Sets the bar number of the trendline start point.
EndBar	Numeric	Sets the bar number of the trendline end point.

Remarks

This function is similar to TLSlope, except that you cannot specify different prices for the start and end points. This function uses the same price for both the start and end bars.

Example

```
Plot1(TLSlopeEasy(High,1,9);
```

See Also

TLSlope

TLValue (Function)

Disclaimer

The TLValue function returns the value of a trendline at specified point.

Syntax

```
  TLValue(StartPrice, StartBar, EndPrice, EndBar, TgtBar)
```

Parameters

Name	Type	Description
StartPrice	Numeric	Sets the price of the trendline start point.
StartBar	Numeric	Sets the bar number of the trendline start point.
EndPrice	Numeric	Sets the price of the trendline end point.
EndBar	Numeric	Sets the bar number of the trendline end point.
TgtBar	Numeric	Set the number of bars to extend trendline (positive number = backwards in time & negative number = forward in time).

Returns (Double)

A numeric value containing the specified trendline at point TgtBar.

Remarks

The two points allow us to calculate the slope of the line; by extending that line, the TLValue function is able to calculate the Price of the target bar in the future.

To calculate the value the function will need the bar numbers and prices for the two endpoints of the trendline as well as the target bar out in the future or back in the past. The input StartPrice and EndPrice are the starting point and ending points of the trendline respectively. They are usually replaced by values such as Close, High, Low, and so on or replaced with numeric series type inputs.

The inputs StartBar and EndBar refer to the bar numbers of the starting and ending points of the trendline. These inputs must be replaced by valid numbers or replaced with numeric simple type inputs.

Note You must offset the price by its corresponding bar number. For example, if you wanted the trendline angle for a line drawn from the Close of ten bars ago to the Close of the current bar, replace the inputs StartPrice, StartBar, EndPrice, and EndBar would be replaced with the values Close[0], 0, Close[10], and 10 respectively.

The input TgtBar must be replaced by either a positive or negative number that represents the number of bars back in the past or out in the future that the trendline is to be extended. Negative values of TgtBar project the trendline into the future, while; positive values of TgtBar project the trendline back into the past

The formula used to calculate the slope is a simple rise over run calculation.

Example

```
  Plot1(TLValue(High,1,High,9,1));
```

See Also

TLValueEasy

TLValueEasy (Function)

Disclaimer

The TLValueEasy function returns the value of a trendline.

Syntax

```
  TLValueEasy(Price, StartBar, EndBar, TgtBar)
```

Returns (Double)

A numeric value containing the specified trendline.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
StartBar	Numeric	Sets the bar number of the trendline start point.
EndBar	Numeric	Sets the bar number of the trendline end point.
TgtBar	Numeric	Set the number of bars to extend trendline (positive number = backwards in time & negative number = forward in time).

Remarks

This function is similar to TLValue, except that you cannot specify different prices for the start and end prices. This function uses the same price on the start and end bar.

The inputs StartBar and EndBar refer to the bar numbers of the starting and ending points of the trendline. These inputs must be replaced by valid numbers or replaced with numeric simple type inputs.

The input TgtBar must be replaced by either a positive or negative number that represents the number of bars back in the past or out in the future that the trendline is to be extended. Negative values of TgtBar project the trendline into the future, while; positive values of TgtBar project the trendline back into the past

Example

```
  Plot1(TLValueEasy(High,1,9,1));
```

See Also

TLValueEasy

TrailingStopOrder (Function)

Disclaimer

The TrailingStopOrder function is used to configure and send a limit if touched order using the order entry macro **.PlaceOrder**. You can call this function directly from your own EasyLanguage code to simplify the formatting and generation of macro orders.

NOTE Care should be exercised when calling this function as it is intended to send live orders. Confirmations for macro-generated orders can be configured by using the File -> Preferences -> TradeStation Order Entry menu sequence.

Syntax

```
TrailingStopOrder(Frequency,Account,Action,SymbolCategory,Symbol,Quantity,Duration,G
TDDate,TrailingAmount,TrailingType)
```

Returns (Integer)

TrailingStopOrder returns 1 if called on a "real-time" tick and if a trade is allowed based on the user input "Frequency". The function returns -1 (negative one) in other cases; for example, if the function is called on an historical bar. If there are errors in the order parameters a runtime error message will be generated and the order will not be placed.

Parameters

Name	Type	Description	Supported Values
Frequency	String	Sets the order frequency.	Once, OncePerBar, EndOfBar, Always
Account	String	Sets the TradeStation account to be used for this order.	*Account string*
Action	String	Sets the order action.	Buy, Sell, SellShort, BuyToCover, BuyToOpen, BuyToClose, SellToOpen, SellToClose
SymbolCategory	String	Sets the trading category for the symbol'.	Equity, Future, Forex, EquityOption, FutureOption
Symbol	String	Sets the symbol to be used for this order.	*Symbol string*
Quantity	Numeric	Sets the number of shares, contracts, lots, etc. to be placed for this order.	Number
Duration	String	Sets the order duration.	Day, Day+, GTC, GTC+, GTD, GTD+, IOC, FOK, OPG, 1Min, 3Min, 5Min
GTDDate	String	Sets the date, if appropriate, to be used with the specified duration.	*Date Format* (MM/DD/YY).
TrailingAmount	Numeric	Sets the trailing amount to be used for this order.	Number
TrailingType	String	Sets the trailing stop calculation to use for this order.	Pts, Pct, Minmove

Remarks

The TrailingStopOrder function uses the PlaceOrder function to format the order parameters and call the order entry macro. The PlaceOrder function is only intended to be used by this and other designated TradeStation functions.

This function disables advanced order placement features (All or None, Buy on minus,Sell on plus, etc.).

See .PlaceOrder command for more information on using the order entry macro for placing orders.

Example

Places a sell limit order of 100 shares for MSFT at a limit price of 24.35 using the order entry macro .PlaceOrder. Value1 returns a 1 if the order is valid.

```
Value1 =
TrailingStopOrder("Once","SIM15180","Sell","Equity","MSFT",100,"Day","",24.65,"MinMo
ve");
```

See Also

LimitIfTouchedOrder, LimitIOrder, MarketIfTouchOrder, MarketOrder, StopLimitOrder, StopMarketOrder, TrailingStopOrder

TriAverage (Function)

Disclaimer

The TriAverage series function places the majority of the weighting on the middle portion of the specified length.

Syntax

```
TriAverage(Price, Length)
```

Returns (Double)

A numeric value containing the Triangular Moving Average for the current bar.

Parameters

Name	Type	Description
`Price`	Numeric	Specifies which bar value (price, function, or formula) to use.
`Length`	Numeric	Sets the number of bars to consider.

Remarks

In calculating the Triangular Average, the Average is inherently double-smoothed.

Examples

Assigns to `Value1` the Triangular Moving Average based on the `Close` of the last 14 bars:

```
Value1 = TriAverage(Close, 14);
```

Assigns to `Value2` the Triangular Moving Average based on the `High` of the last 21 bars.

```
Value2 = TriAverage(High, 21);
```

Additional Example

If you wanted to place a Buy on `Close` order when the `Close` crossed above the Triangular Moving Average, you could use the following syntax:

```
If Close Crosses Above TriAverage(Close, 18) Then
 Buy This Bar on Close;
```

See Also

Average, WAverage, and XAverage

TriAverage_Gen (Function)

Disclaimer

The TriAverage_Gen series function returns a smoothed average that places the majority of the weighting on the middle portion of the specified length.

Syntax

```
TriAverage_Gen(Price, Length)
```

Returns (Double)

The function returns the triangular average value.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Length	Numeric	Sets the number of bars to consider.

Remarks

This function returns the triangular average of values over all Length bars compared to the TriAverage function that returns the triangular average of an even number of Length bars.

In calculating the Triangular Average, the Average is inherently double-smoothed.

The input parameter Price can be a bar value such as Close, High, Low, Open, or Volume. It can also be any mathematical calculation such as: (High + Low) / 2, or a numeric function such as RSI, Stochastic, or ADX.

Examples

Assigns to Value1 the Triangular Moving Average based on the Close of the last 14 bars:

```
Value1 = TriAverage_Gen(Close, 14);
```

Assigns to Value2 the Triangular Moving Average based on the High of the last 21 bars.

```
Value2 = TriAverage_Gen(High, 21);
```

Additional Example

If you wanted to place a Buy on Close order when the Close crossed above the Triangular Moving Average, you could use the following syntax:

```
If Close Crosses Above TriAverage_Gen(Close, 15) Then
 Buy This Bar on Close;
```

TrimMean (Function)

Disclaimer

The TrimMean function calculates the Interior Mean value of the specified bar after trimming extremes.

Syntax

```
TrimMean(Price, Length, TrimPct)
```

Returns (Double)

A numeric value containing the mean of a range of bars. If TrimPct is less than 0, or TrimPct is greater than 1, then the Function will return a -1.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Length	Numeric	Sets the number of bars used in the calculation.
TrimPct	Numeric	Sets the percentage of data to exclude from calculation. Enter 0.20 for 20% which will exclude 10% of the data values from each end of the number of bars selected.

Remarks

TrimMean calculates the mean taken by excluding a percentage (TrimPct) of data points from the top and bottom tails of a data set. This function can be used to exclude outlying data from your analysis.

If TrimPct is less than 0 or if TrimPct is greater than 1, the function will return -1

Examples

Assigns to Value1 the TrimMean value calculated over the past 14 closing prices with a TrimPct of .25 or 25%:

```
Value1 = TrimMean(Close, 14, .25);
```

Assigns to Value2 the TrimMean value calculated over the past 14 High prices with a TrimPct of .10 or 10%:

```
Value2 = TrimMean(High, 21, .1);
```

TrimMeanArray (Function)

Disclaimer

The TrimMeanArray function calculates the mean value of the specified array after trimming away a percentage of values at each end of the array.

Syntax

```
TrimMeanArray(PriceArray, Size, TrimPct)
```

Returns (Double)

A numeric value containing the mean of the interior of the array. If TrimPct is less than 0, or TrimPct is greater than 1, then the Function will return a -1.

Parameters

Name	Type	Description
PriceArray	Numeric Array	Specifies the name of a numeric array containing bar values used to calculate the TrimMean value.
Size	Numeric	Sets the number of data elements (size) in the array to evaluate.
TrimPct	Numeric	Sets the percentage of data to be excluded from both ends of an array (enter .25 for 25%).

Remarks

TrimMeanArray calculates the mean taken by excluding a percentage "TrimPct" of data points from the extreme tails of the array. This function can be used to exclude outlying data from your analysis.

The value for the Size input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The TrimMeanArray function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Example

The following assigns the lowest value of the 20 elements in the array to Value1:

```
Array: myArray[20](0);
{… add EL statements here to assign values to array elements... }
Value1 = TrimMeanArray(MyArray, 20);
```

See Also

ExtremesArray

TRIX (Function)

Disclaimer

The TRIX function calculates the percent rate-of-change of a triple exponentially smoothed moving average of the security's closing price.

Syntax

```
TRIX(Price, Length)
```

Returns (Double)

A numeric value containing TRIX for the current bar.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Length	Numeric	Sets the number of bars to consider.

Remarks

TRIX is a momentum indicator that displays the percent rate-of-change of a triple exponentially smoothed moving average of the security's closing price. The TRIX indicator oscillates around a zero line. Its triple exponential smoothing is designed to filter out noise.

Example

If you want to initiate a short position when TRIX crosses under zero you can write:

```
If TRIX(Close, 10) Crosses Under 0 Then
 SellShort Next Bar at Market;
```

TrueHigh (Function)

Disclaimer

The TrueHigh function returns either the High of the current bar or the Close of the previous bar if its value is higher.

Syntax

```
TrueHigh
```

Returns (Double)

A numeric value containing the greater of the current bar High or previous bar Close.

Parameters

None

Remarks

The TrueHigh function is most commonly used in finding Gap Open bars. A Gap Open bar is when the Open is greater than the previous bars High.

Example

```
Plot1(TrueHigh, "TrueHi");
```

See Also

TrueLow, TrueRange

TrueLow (Function)

Disclaimer

The TrueLow function returns either the Low of the current bar or the Close of the previous bar if its value is lower.

Syntax

```
TrueLow
```

Returns (Double)

A numeric value containing the lesser of the current bar Low or previous bar Close.

Parameters

None

Example

```
Plot1(TrueLow, "TrueLo");
```

See Also

TrueHigh, TrueRange

TrueRange (Function)

Disclaimer

The TrueRange function returns the difference between the TrueHigh and TrueLow values.

Syntax

```
TrueRange
```

Returns (Double)

A numeric value containing the difference between the TrueHigh and TrueLow for the current bar.

Parameters

None

Remarks

This function is similar to Range except that it uses the TrueHigh and TrueLow values that take in to account the previous bar close in addition to the current bar High and Low.

TrueRange is defined as the larger of the following:

- The distance between today’s High and today's Low.
- The distance between today’s High and yesterday's Close.
- The distance between today’s Low and yesterday's Close.

Example

```
Plot1(TrueRange, "TRange");
```

TrueRangeCustom (Function)

Disclaimer

The TrueRangeCustom function calculates the true range based on user specified high and low prices.

Syntax

```
TrueRangeCustom(HPrice, LPrice, CPrice)
```

Returns (Double)

A numeric value represent the true range based on a user's set of prices.

Parameters

Name	**Type**	**Description**
PriceH	Numeric	Sets a user specified high price as the basis for the true range calculations.
PriceL	Numeric	Sets a user specified low price as the basis for the true range calculations.
PriceC	Numeric	Specifies which bar value (price, function, or formula) to use for the close.

Remarks

This function is similar to TrueRange except that it allows greater flexibility through the use of user specified high and low price inputs.

Example

Assigns to Value1 the 'true range' of prices that are outside of the current bars actual range.

```
Value1 = TrueRangeCustom(High+.50, Low-.50, Close);
```

TypicalPrice (Function)

Disclaimer

The `TypicalPrice` function returns average of the `High`, `Low` and `Close` prices of a bar.

Syntax

```
TypicalPrice
```

Returns (Double)

A numeric value containing the typical average price for the current bar.

Parameters

None

Example

Assigns to `Value1` the typical average price on a bar, then plots `Value1`.

```
Value1 = TypicalPrice;
Plot1(Value1, "AvgPrc");
```

UlcerIndex (Function)

Disclaimer

The `UlcerIndex` function is a measure of the stress level related to market conditions.

Function

```
UlcerIndex(Price, Length)
```

Returns (Double)

A numeric value containing the Ulcer Index for the current bar.

Parameters

Name	Type	Description
`Price`	Numeric	Specifies which bar value (price, function, or formula) to be considered.
`Length`	Numeric	Sets the number of bars to consider.

Remarks

The Ulcer Index uses retracements to measure the "stressfulness" associated with the instrument at the current time. As stated by Peter Martin and Byron McCann, "the higher an investment's UI, the more likely investing in it will cause ulcers or sleepless nights."

Example

Assigns to `Value1` the Ulcer Index calculated over the past 14 closing prices:

```
Value1 = UlcerIndex(Close, 14);
```

Assigns to `Value2` the Ulcer Index calculated over the past 21 `High` prices:

```
Value2 = UlcerIndex(High, 21);
```

UltimateOscillator (Function)

Disclaimer

The `UltimateOscillator` function returns the Ultimate Oscillator value developed by Larry Williams.

Function

```
UltimateOscillator(ShortLength, MiddlLength, LongLength)
```

Returns (Double)

A numeric value containing the current value of the Ultimate Oscillator.

Parameters

Name	Type	Description
ShortLength	Numeric	Sets the number of bars in short term average.
MiddlLength	Numeric	Sets the number of bars in intermediate term average.
LongLength	Numeric	Sets the number of bars in long term average.

Remarks

Williams contends that an oscillator based on one time span is subject to a number of false alerts. To combat this problem, he combines three such oscillators, each based on different time frames. Williams states that oscillators with shorter time frames tend to peak well ahead of price, usually causing several divergences prior to the ultimate peak. On the other hand, indicators based on longer time spans do not tend to reverse direction until after market turning points.

Example

Assigns to Value1 and plots the `UltimateOscillator` based on the specified short, medium, and long term averages.

```
Value1 = UltimateOscillator(7,14,28);
Plot1(Value1, "UltOsc:);
```

Reference

Larry Williams

VarianceArray (Function)

Disclaimer

The `VarianceArray` function calculates the estimated variance for an array of values.

Syntax

```
VarianceArray(PriceArray, Size, DataType)
```

Returns (Double)

A numeric value containing the estimated variance based either on the population or a sample of an array, as determined by the specified inputs.

Parameters

Name	Type	Description
`PriceArray`	Numeric	Specifies the name of a numeric array containing values upon which the variance is calculated.
`Size`	Numeric	Sets the number of data elements (size) in the array to evaluate.
`DataType`	Numeric	Sets what data elements should be based on: 1 = population, 2 = sample

Remarks

Variance is the measure of the "scatter" of prices around the mean of a sample.

The value for the `Size` input parameter should always be a whole number greater than 0 and is typically equal to the number of data elements in the array.

The `VarianceArray` function only works with one-dimensional arrays. All array-based referencing begins with array element 1.

Note This array function is similar to `Variance` except that you can build an array of values that does not necessarily include contiguous bar values.

Example

Assigns to `Value1` the variance for the named array for all 10 elements in the array .

```
Array: myArray[10](0);
{… (assign values to array) }
Value1 = VarianceArray (myArray, 10);
```

VariancePS (Function)

Disclaimer

The VariancePS function calculates the estimated variance based either on the population or a sample.

Syntax

```
VariancePS(Price, Length, DataType)
```

Parameters

Name	Type	Description
`Price`	Numeric	Specifies which bar value (price, function, or formula) to consider.
`Length`	Numeric	Sets the number of trailing bars to consider.
`DataType`	Numeric	Sets the data type. 1 = population, 2 = sample

Returns (Double)

A numeric value containing the estimated variance for a set of values over length bars.

Remarks

Variance is the measure of the "scatter" of prices around the mean of a sample.

Example

Assigns to `Value1` the Variance calculated over the past 14 closing prices.

```
Value1 = VariancePS(Close, 14, 1);
```

Assigns to `Value2` the sample Variance calculated over the past 21 `High` prices.

```
Value2 = VariancePS(High, 21, 2)
```

Volatility (Function)

Disclaimer

The `Volatility` series function measures the market volatility by plotting a smoothed average of the `TrueRange`. It returns an average of the `TrueRange` over a specific number of bars, giving higher weight to the `TrueRange` of the most recent bar.

Syntax

```
Volatility(Length)
```

Returns (Double)

A numeric value containing the market volatility. As the number increases, the market is more volatile.

Parameters

Name	Type	Description
`Length`	Numeric	Sets the number of bars to include in the volatility calculation.

Remarks

Volatility is the variation in price over a specific interval (the difference between the highest and lowest prices). As the time interval being studied increases, volatility also increases to a maximum before leveling off. As prices increase, the volatility tolerance also increases.

Note The `Volatility` function uses a slightly different set of calculations than the original formula. The Volatility variation tends to smooth recent activity, which means that it will take more time (bars) to 'normalize.' The original formula is provided in the `VolatilityClassic` function.

Example

Plots the volatility over the last 20 bars.

```
Plot1(Volatility(20));
```

Assigns to Value1 the volatility of the last 15 bars.

```
Value1 = Volatility(15):
```

Reference

Kaufman, P.J. *The New Commodity Trading Systems and Methods*. John Wiley & Sons. New York 1980. Pages 99-101.

VolatilityStdDev (Function)

Disclaimer

The VolatilityStdDev function calculates the statistical volatility, sometime called historical volatility, based on a standard deviation of closes for the number of days specified.

Syntax

```
VolatilityStdDev(NumDays)
```

Returns (Double)

A numeric value containing VolatilityStdDev, based on a standard deviation of closes for the number of days specified.

Parameters

Name	Type	Description
NumDays	Numeric	Sets the number of bars to include in the volatility calculation

Remarks

The statistical volatility is sometimes called historical volatility. The volatility returned is annualized and is an indication of how much price movement as a percent has been observed over the specified time period.

Example

Assigns to Value1 the annualized statistical volatility for the last 30 days of closing prices for the symbol using the Standard Deviation of Closes Method.

```
Value1 = VolatilityStdDev(30);
```

VolumeOsc (Function)

Disclaimer

The `VolumeOsc` function calculates the difference between a slow and fast period moving average in terms of points.

Syntax

```
VolumeOsc(FastLength, SlowLength)
```

Returns (Double)

A numeric value containing a positive or negative value of `VolumeOsc` for the current bar.

Parameters

Name	Type	Description
`FastLength`	Numeric	Sets the time period of the slow moving average.
`SlowLength`	Numeric	Sets the time period of the fast moving average.

Example

```
Value1 = VolumeOsc(9,18);
```

VolumeROC (Function)

Disclaimer

The VolumeROC function returns the volume rate of change based on a range of bars..

Syntax

```
VolumeROC(Length)
```

Returns (Double)

A numeric value containing a positive or negative value of VolumeROC for the current bar.

Parameters

Name	Type	Description
Length	Numeric	Sets the number of bars to consider.

Remarks

The most common use of this function is to determine the likelihood of a continuation in the current move.

Example

```
Value1 = VolumeROC(20);
```

WAverage (Function)

Disclaimer

The WAverage is a weighted moving average of the Price over the Length specified. It is calculated by giving the more current data a heavier weight and the oldest data less weight.

Syntax

```
WAverage(Price, Length)
```

Returns (Double)

A numeric value containing the weighted moving average over a specified number of bars.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Length	Numeric	Sets the number of bars to consider.

Remarks

Just as the Average function needs a Price and Length to calculate, the WAverage function also needs the Price and Length inputs. The Price is the unit you want to average, and the Length is the number of units you are averaging. The WAverage is used in the same way as the average function and can give stronger and earlier indications to trend direction as it follows the most recent data closer.

Example

```
Plot1(WAverage(Close,20));
```

See Also

Average, TriAverage, and XAverage

WeightedClose (Function)

Disclaimer

The WeightedClose function returns an average price that gives more precedence to the Close than the High or Low.

Syntax

```
WeightedClose
```

Returns (Double)

A numeric value containing the weighted close.

Parameters

None

Remarks

The WeightedClose function calculates the average price of the bar by taking the High and Low and adding it to two Closes and dividing by four.

Example

```
Value1 = WeightedClose;
```

XAverage (Function)

Disclaimer

The XAverage series function is an exponentially weighted moving average of the prices of the last length bars.

Syntax

```
XAverage(Price, Length)
```

Returns (Double)

A numeric value containing the exponential average over a specified number of bars.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Length	Numeric	Sets the number of bars to consider.

Remarks

When this moving average is calculated, every bar's price in the data file will have an effect on the current average. The effects of the first price will never be completely removed, but its weight will continuously shrink as more and more averages are calculated.

Example

```
Plot1(XAverage(Close,20));
```

Reference

Kaufman, P.J. *The New Commodity Trading Systems and Methods*. John Wiley & Sons. New York 1980. Page 40.

See Also

Average, TriAverage, and WAverage

XAverageOrig (Function)

Disclaimer

The XAverageOrig series function is an exponentially weighted moving average of the prices of the last length bars using an alternate smoothing factor from XAverage.

Syntax

```
XAverageOrig(Price, Length)
```

Returns (Double)

A numeric value containing the exponential average over a specified number of bars.

Parameters

Name	Type	Description
Price	Numeric	Specifies which bar value (price, function, or formula) to use.
Length	Numeric	Sets the number of bars to consider.

Remarks

When this moving average is calculated, every bar's price in the data file will have an effect on the current average. The effects of the first price will never be completely removed, but its weight will continuously shrink as more and more averages are calculated.

XAverageOrig differs from XAverage due to the smoothing factor. The smoothing factor in XAverage is 2 / (Length + 1) and the smoothing factor in XAverageOrig is the reciprocal of the number of bars to consider, 1/Length.

Example

```
Plot1(XAverageOrig(Close,20));
```

Reference

Kaufman, P.J. *The New Commodity Trading Systems and Methods*. John Wiley & Sons. New York 1980. Page 40.

See Also

XAverage

ZProb (Function)

Disclaimer

The `ZProb` function returns the two-tailed P-value of a Z-Test for the specified bar.

Syntax

```
ZProb(Price, Length)
```

Returns (Double)

A numeric value containing the two-tailed P-value of a Z-Test for the specified bar.

Parameters

Name	Type	Description
`Price`	Numeric	Specifies which bar value (price, function, or formula) to use.
`Length`	Numeric	Sets the number of bars to consider.

Remarks

The z-test generates a standard score for x with respect to the data set (`Price`), array, and returns the two-tailed probability for the normal distribution.

The function can be used to assess the likelihood that a particular observation is drawn from a particular population.

Example

Assigns to **`Value1`** the Z-Test value based on the closing prices of the data stream:

```
Value1 = ZProb(Close, 14);
```

Assigns to `Value2` the Z-Test value based on the TrueRange of the data stream:

```
Value2 = ZProb(TrueRange, 5);
```

Reserved Words

#BEGINALERT (Reserved Word)

Disclaimer

The statements between this compiler directive and the reserved word `#End` are evaluated only when the alert is enabled for the analysis technique. The reserved word `#End` must be must be used with this reserved word.

Note All statements between `#BeginAlert` and `#End` are ignored, including the calculation of MaxBarsBack, unless the **Enable Alert** check box is selected.

Example

```
#BeginAlert
If Close[50] > Close and ADX(14) > ADX(14)[1] then
 Alert("ADX Alert");
#End;
```

Additional Example

An indicator that calculates the 10-bar momentum of the closing price needs ten bars in order to start plotting results. If an alert is added to this indicator and the alert uses a 50-bar average of the volume, then the bar requirement is upped to fifty. However, the 50-bar average is only used for the alert calculations, so there is no need to have the indicator *wait* fifty bars before returning results unless the alert is enabled.

Therefore, to have the indicator plot results after ten bars and ignore the 50-bar requirement, use `#BeginAlert` in your indicator, as follows:

```
Plot1(Close - Close[10], "Momentum");
#BeginAlert ;
 If Plot("Momentum") Crosses Over 0 AND Volume > Average(V, 50)* 2 Then
  Alert("Momentum is now positive") ;
#End ;
```

The above indicator plots the momentum and triggers an alert if the momentum becomes positive while experiencing volume that is greater than twice the 50-bar average. When the indicator is applied without enabling the alert, it requires only ten bars to start calculating. When the alert is enabled, the indicator is recalculated; the statements within the compiler directives are evaluated and the new requirement is 50 bars.

#BEGINCMTRY (Reserved Word)

Disclaimer

The statements between this compiler directive and the reserved word #End are evaluated only when the **Analysis 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.

Note All statements between the #BeginCmtry and #End are ignored, including calculation of MaxBarsBack, unless commentary is generated.

Example

```
#BeginCmtry
 Commentary("The indicator value here is " + NumtoStr(Plot1, 2));
#End;
```

Additional 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 Plot("Momentum") > 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.

#BEGINCMTRYORALERT (Reserved Word)

Disclaimer

The statements between this compiler directive that and the reserved word #End are evaluated only when the **Enable Alert** check box is selected when using the **Analysis Commentary** tool to select a bar on a chart or a cell in a grid. The reserved word #End must be used with this reserved word.

Note All statements between #BeginCmtryOrAlert and #End are ignored, including calculation of MaxBarsBack, unless the **Enable Alert** check box is selected or commentary is generated.

Example

The following indicator plots the 10-bar momentum of the close, and triggers an alert when the momentum becomes positive while experiencing volume that is greater than twice the 50-bar average. In addition, commentary is written to help point out the market conditions bar by bar.

```
Plot1(Close - Close[10], "Momentum");

#BeginCmtryOrAlert ;

 If Plot("Momentum") > 0 Then
  Commentary("Momentum is positive, ")
 Else
  Commentary("Momentum is negative, ");
 If Volume > Average(Volume, 50) Then Begin
  Commentary("and volume is greater than average.");
  If Volume > Average(Volume, 50) * 2 Then
   Alert;
 End
 Else
  Commentary("and volume is lower than average.");

#End ;
```

#END (Reserved Word)

Disclaimer

A compiler directive used in conjunction with #BeginAlert, #BeginCmtry, and #BeginCmtryorAlert to terminate an alert or commentary statement.

Example

```
#BeginCmtryOrAlert
 If Close[50] > Close and ADX(14) > ADX(14)[1] then
  Alert("ADX Alert");
 Commentary("The indicator value here is " + NumtoStr(Plot1, 2));
#End;
```

#EVENTS (Reserved Word)

Disclaimer

#EVENTS is a compiler directive used to declare an event block. See #END.

A (Reserved Word)

Disclaimer

A is a Skip Word that performs no function within your code. It is used within a statement only to make the EasyLanguage code more readable.

Remarks

Skip words in EasyLanguage are not necessary, however, they sometimes make it easier to understand the purpose of the code.

Syntax Coloring in the EasyLanguage PowerEditor is dark green for all skip words by default.

Example

```
If a Close is > 100 Then
      Alert;
```

The word "a" is not necessary and the code functions the same way regardless of its presence.

AB_AddCell (Reserved Word)

Disclaimer

This reserved word is used to add a cell to the current bar of the chart. You can only add cells to the bar currently being analyzed (for example, `AB_AddCell(...)[1]` is not allowed). This re served word must be included in an ActivityBar study.

```
   AB_AddCell(Price, Side, Str_Char, Color, Value)
```

`Price` is a numeric expression representing the price value at which the cell is added. It can be any value inside or outside the range of the bar.

`Side` specifies the side of the bar on which the cell is placed, and it accepts one of two reserved words, `LeftSide` or `RightSide`.

`Str_Char` is the text string expression representing the text stored in the cell being added. The expression is limited to one character. If the text string expression is longer than one character, only the first character is used (for example, if the text string expression is "High", the letter "H" is placed in the cell).

`Color` is the EasyLanguage color or its numeric equivalent representing the color in which the cell is drawn.

`Value` is a numeric expression stored in the cell. This value is required; however, it does not affect the calculation of the ActivityBar study. The value can be referred to later from the ActivityBar study itself or from other analysis techniques that make reference to the ActivityBar study. Use zero (0) for this parameter if you do not want to specify a meaningful value.

Notes

When writing an ActivityBar study, you must also specify the cell height. To do so, use the reserved word `AB_SetRowHeight`.

Example

The following statement adds a green cell to the right side of the bar for every tick. Each cell contains the letter A, and stores the trade volume for the tick in each cell:

```
   AB_AddCell(Close of ActivityData, RightSide, "A", Green, Volume of ActivityData);
```

AB_GetCellChar (Reserved Word)

Disclaimer

This reserved word returns the text string expression held by the specified cell.

```
AB_GetCellChar(Price, Side, Offset)
```

Price is a numeric expression representing the price of the cell referenced. Side specifies the side of the bar on which the cell is located (you must use one of two reserved words, LeftSide or RightSide, to specify the side), and Offset is the column number of the cell referenced (where column 1 is the closest to the bar).

Remarks:

The specified cell must be applied to the chart to return a value. If you reference a cell that does not exist, a runtime error will occur.

Notes

You can use this reserved word in any analysis technique, trading strategy, or function. To store the text string expression returned by the reserved word, you can assign this reserved word to a text string variable.

Example

```
AB_GetCellChar(Close, Rightside, 3);
```

Returns the character string ("2" for example) from the ActivityBar cell 3 cells to the right of the bar at price, close.

Example

The following statements retrieve the text string expression held in the first cell of the row corresponding to the closing price of the current bar:

```
Variable: Str(" ");
Str = AB_GetCellChar(Close of Data1, LeftSide, 1);
```

AB_GetCellColor (Reserved Word)

Disclaimer

This reserved word returns a number representing the color used to draw the specified cell.

```
AB_GetCellColor(Price, Side, Offset)
```

`Price` is a numeric expression representing the price of the cell referenced. `Side` specifies the side of the bar on which the cell is located (you must use one of two reserved words, `LeftSide` or `RightSide`, to specify the side), and `Offset` is the column number of the cell referenced (where column 1 is the closest to the bar).

Notes

To store the number returned by the reserved word, you can assign this reserved word to a numeric variable. The numeric value returned is the EasyLanguage numeric equivalent used to specify colors. You can use this reserved word in any analysis technique, trading strategy, or function. If you reference a cell that does not exist, a runtime error will occur.

Example

```
GetCellColor(Open, Leftside, 3);
```

Returns the numeric color value (7 for example if the cell is yellow) from the ActivityBar cell 3 cells to the left of the bar at the open price.

Additional Example

The following statement retrieves the color of the first cell on the right side located at the opening price of the bar. The color is assigned to the variable `Value1`:

```
Value1 = AB_GetCellColor(Open of Data1, RightSide, 1);
```

AB_GetCellDate (Reserved Word)

Disclaimer

Each time a cell is added to a bar, the date and time of when it was added is stored with the cell. This reserved word returns the EasyLanguage date corresponding to the date the cell was added to the bar.

```
AB_GetCellDate(Price, Side, Offset)
```

`Price` is a numeric expression representing the price of the cell being referenced. `Side` spec ifies the side of the bar on which the cell is located (you must use one of two reserved words, `LeftSide` or `RightSide`, to specify the side), and `Offset` is the column number of the cell referenced (where column 1 is the closest to the bar).

Notes

To store the date returned by the reserved word, you can assign this reserved word to a numeric variable. You can use this reserved word in any analysis technique, trading strategy, or function. If you reference a cell that does not exist, a runtime error will occur.

Example

```
AB_GetCellDate(Open, Leftside, 3);
```

Returns the date value (990115 for example if the date of the cell is January 15, 1999) from the ActivityBar cell 3 cells to the left of the bar at the open price.

Additional Example

The following statement retrieves the date of the first cell on the right side at the opening price of the bar, and assigns this date to the numeric variable `Value1`:

```
Value1 = AB_GetCellDate(Open of Data1, RightSide, 1);
```

AB_GetCellTime (Reserved Word)

Disclaimer

Each time a cell is added to a bar, the date and time of when it was added is stored with the cell. This reserved word returns the time that the cell was added to the bar.

```
AB_GetCellTime(Price, Side, Offset)
```

`Price` is a numeric expression representing the price of the cell being referenced. `Side` spec ifies the side of the bar on which the cell is located (you must use one of two reserved words, `LeftSide` or `RightSide`, to specify the side), and `Offset` is the column number of the cell referenced (where column 1 is the closest to the bar).

Notes

To store the time returned by the reserved word, you can assign this reserved word to a numeric variable. You can use this reserved word in any analysis technique, trading strategy, or function. If you reference a cell that does not exist, a runtime error will occur.

Example

```
AB_GetCellTime(Close, RightSide, 3);
```

Returns the time value (1025 for example if the time of the cell is 10:25am) from the ActivityBar cell 3 cells to the right of the bar at the close price.

Additional Example

The following statement retrieves the time of the first cell on the right side at the opening price of the bar, and assigns this date to the numeric variable `Value1`:

```
Value1 = AB_GetCellDate(Open of Data1, RightSide, 1);
```

AB_GetCellValue (Reserved Word)

Disclaimer

When you add a cell to a bar using the `AB_AddCell` reserved word, you can store a value in the cell. You use the `AB_GetCellValue` reserved word to obtain the value.

```
AB_GetCellValue(Price, Side, Offset)
```

`Price` is a numeric expression representing the price of the cell being referenced. `Side` spec ifies the side of the bar on which the cell is located (you must use one of two reserved words, `LeftSide` or `RightSide`, to specify the side), and `Offset` is the column number of the cell referenced (where column 1 is the closest to the bar).

Notes

To store the value returned by the reserved word, you can assign this reserved word to a numeric variable. You can use this reserved word in any analysis technique, trading strategy, or function. If you reference a cell that does not exist, a runtime error will occur.

Example

```
AB_GetCellValue(Close, Rightside, 3);
```

Returns the value (100 for example if the value of the cell is 100) from the ActivityBar cell 3 cells to the right of the bar at the close price.

Additional Example

The following statement retrieves the value stored in the first cell on the right side at the opening price of the bar, and assigns this value to the numeric variable `Value1`:

```
Value1 = AB_GetCellValue(Open of Data1, RightSide, 1);
```

AB_GetNumCells (Reserved Word)

Disclaimer

This reserved word returns the number of cells in a specified row.

```
AB_GetNumCells(Price, Side)
```

`Price` is a numeric expression representing the price at which you want to place the marker, and `Side` defines the marker to move (left or right). `Side` only accepts one of two reserved words, `LeftSide` or `RightSide`.

Notes

If you reference any attribute of a non-existent cell, a run time error is generated by the ActivityBar study when applied to a chart. You can avoid these errors by using the `AB_GetNumCells` reserved word to determine the number of available cells before attempt ing to reference any of them.

To store the resulting value, you can assign this reserved word to a numeric variable. You can use this reserved word in any analysis technique, trading strategy, or function.

Example

```
AB_GetNumCells(Open, Leftside);
```

Returns the number of ActivityBar cells to the left of the bar at price, open.

Additional Example

The following statements obtain the text string expression stored in the last cell in the row corresponding to the open of the bar. Notice that we first obtain the total number of cells in the desired row, and store this number in the variable `Value1`. We then use the resulting number (`Value1`) to obtain the text string expression:

```
Variable: Str(" ");
Value1 = AB_GetNumCells(Open of Data1, RightSide);
Str = AB_GetCellChar(Open of Data1, Value1, RightSide);
```

AB_GetZoneHigh (Reserved Word)

Disclaimer

This reserved word returns a numeric value representing the upper boundary of the ActivityBar study zone.

```
AB_GetZoneHigh(Side)
```

`Side` specifies the side for which to obtain the value. *Side* accepts one of two reserved words, `LeftSide` or `RightSide`.

Notes

To store the resulting value, you can assign this reserved word to a numeric variable. You can use this reserved word in any analysis technique, trading strategy, or function.

Example

The following statement assigns the high price of the ActivityBar zone to the numeric variable `Value1`:

```
Value1 = AB_GetZoneHigh(RightSide);
```

AB_GetZoneLow (Reserved Word)

Disclaimer

This reserved word returns a numeric value representing the lower boundary of the ActivityBar study zone.

```
AB_GetZoneLow(Side)
```

`Side` specifies the side for which to obtain the value. *Side* accepts one of two reserved words, `LeftSide` or `RightSide`.

Notes

To store the resulting value, you can assign this reserved word to a numeric variable. You can use this reserved word in any analysis technique, trading strategy, or function.

Example

The following statement assigns the low price of the ActivityBar zone to the numeric variable `Value1`:

```
Value1 = AB_GetZoneHigh(RightSide);
```

AB_High (Reserved Word)

Disclaimer

This reserved word returns a numeric value representing the highest price on the bar at which a cell is drawn.

```
AB_High
```

Notes

If no cells are drawn, a value of zero (0) is returned. To store the resulting value, you can assign this reserved word to a numeric variable. You can use this reserved word in any analysis technique, trading strategy, or function.

Example

```
Value1 = AB_High;
```

Sets the value of a variable, `Value1`, to the high value of current ActivityBar.

Additional Example

The following statements use a `While` loop to traverse all the possible cells:

```
Value1 = AB_High;

While Value1 > AB_Low Begin
 { EasyLanguage Instruction(s) }
 Value1 = Value1 - AB_GetRowHeight;
End;
```

First, we use `AB_High` to obtain the highest price at which a cell is drawn, and we assign this value to `Value1`. In each iteration of the While loop, we subtract the value equal to one row (which we obtain using `AB_GetRowHeight`). The loop continues as long as `Value1` is greater than the lowest price on the bar at which a cell is drawn.

AB_Low (Reserved Word)

Disclaimer

This reserved word returns a numeric value representing the lower of two values: the lowest price of the bar on which the ActivityBar study is applied, or the lowest price on the bar at which a cell is drawn.

```
AB_Low
```

Notes

If no cells are drawn, a value of zero (0) is returned. To store the resulting value, you can assign this reserved word to a numeric variable. You can use this reserved word in any analysis technique, trading strategy, or function.

Example

```
Value1 = AB_Low;
```

Sets the value of a variable, `Value1`, to the low value of current ActivityBar.

Example

The following statements use a `While` loop to traverse all the possible cells:

```
Value1 = AB_Low;

While Value1 < AB_High Begin
 { EasyLanguage Instruction(s) }
 Value1 = Value1 + AB_GetRowHeight;
End;
```

First, we use `AB_Low` to obtain the lowest price at which a cell is drawn, and we assign this value to `Value1`. In each iteration of the `While` loop, we add the value equal to one row (which we obtain using `AB_GetRowHeight`). The loop continues as long as `Value1` is less than the highest price on the bar at which a cell is drawn.

AB_RemoveCell (Reserved Word)

Disclaimer

This reserved word is used to remove a cell from the current bar of an ActivityBar study.

```
AB_RemoveCell(Price, Offset, Side)
```

`Price` is a numeric expression representing the price of the row from which the cell is to be removed. `Offset` is the column number of the cell to be removed (where column 1 is the closest to the bar), and `Side` specifies the side of the bar on which the cell is located (you must use one of two reserved words, `LeftSide` or `RightSide`, to specify the side).

Notes

If the specified cell does not exist, the ActivityBar study generates a run time error with the message "ActivityBar tried to reference an empty row."

Example

Removes the ActivityBar cell 2 cells to the right of the bar at the closing price:

```
AB_RemoveCell(Close, 2, RightSide)
```

Additional Example

The following statement removes the last cell on the right side of the bar, from the row cor responding to the close of the bar:

```
Value1 = AB_GetNumCells(Close of Data1, RightSide);
AB_RemoveCell(Close of Data1, Value1, RightSide);
```

This example uses the reserved word `AB_GetNumCells` to obtain the number of cells on the right side of the ActivityBar, and then uses the value obtained as the `Offset` parameter for `AB_RemoveCell`.

AB_RowHeight (Reserved Word)

Disclaimer

Returns the row height value of ActivityBar data.

```
AB_RowHeight
```

Remarks

The value returned will be based on the row height of the ActivityBar study and can be set using the SetRowHeight Reserved Word.

Example

```
   Value1 = AB_RowHeight;
```

Sets the value of a variable, `Value1`, to the row height value of the ActivityBar study.

AB_SetActiveCell (Reserved Word)

Disclaimer

ActivityBar studies display price markers on each bar on the chart. By default, these markers are drawn at the open (left side) and closing prices (right side). This reserved word overrides the default placement of these markers, allowing you to place them at any location on the bar.

```
AB_SetActiveCell(Price, Side)
```

`Price` is a numeric expression representing the price at which you want to place the marker, and `Side` defines the marker to move (left or right). `Side` only accepts one of two reserved words, `LeftSide` or `RightSide`.

Example

Sets the row of cells to the right of the bar at the closing price to active status.

```
AB_SetActiveCell(Close, RightSide);
```

Additional Example

The following statements place the right side marker at the modal cell of the ActivityBar study:

```
Value1 = AB_Mode(RightSide);
AB_SetActiveCell(Value1, RightSide);
```

AB_SetRowHeight (Reserved Word)

Disclaimer

This reserved word is used to define the height of each cell (row) on a bar-by-bar basis; it is required when writing an ActivityBar study.

```
AB_SetRowHeight(Value)
```

`Value` is a numeric expression representing the row height.

Notes

You want the row height to be dynamic because symbols vary greatly in price from one symbol to another. For example, a row height of 0.25 will work nicely if the instrument is trading at $50, but it will be an enormous row height for a penny stock trading at $1 per share. Also, the trading range for a symbol can change significantly during a span of several years (for example, a stock adjusted for several stock splits), and an appropriate row height for to day may not work well in the past or the future. The built-in ActivityBar studies use the reserved word `AB_RowHeightCalc` as the parameter for this reserved word to calculate a dynamic row height.

When writing an ActivityBar study, you must also use the `AB_AddCell` reserved word to add cells.

Example

The following statement sets the row height to 1/20th of the average range of the last 10 bars. The result is approximately 20 rows of cells per bar:

```
AB_SetRowHeight(Average(Range, 10) / 20 );
```

AB_SetZone (Reserved Word)

Disclaimer

This reserved word defines the properties of the ActivityBar study zone.

```
AB_SetZone(HighVal, LowVal, Side)
```

`HighVal` and `LowVal` are numeric expressions representing the upper and lower boundaries of the ActivityBar study zone, respectively. `Side` is one of two reserved words `LeftSide` or `RightSide`, which specifies the side of the bar on which the zone is drawn.

Notes

The zone is drawn on every bar using the same drawing properties (color and thickness) of the bars, and is wide enough to fit the widest row of cells of that bar. The ActivityBar study zone is not drawn if there are no cells for a bar.

Example

Sets the zone range box with a high level of the 5 bar moving average of the high prices, a low level of the 5 bar moving average of the low prices, and places it on the right side of the bar.

```
AB_SetZone(Average(High, 5), Average(Low, 5), RightSide);
```

Additional Example

The following statements draw the ActivityBar study zone to the left of each bar at one standard deviation above and below the median price of the ActivityBar cells:

```
Value1 = AB_Median(RightSide);
Value2 = AB_StdDev(1, RightSide);

AB_SetZone(Value1 + Value2, Value1 - Value2, RightSide);
```

Above (Reserved Word)

Disclaimer

This word is used to check for the direction of a cross between values.

`Above`

Remarks

Used in conjunction with the reserved word "`Crosses`" to detect when a value crosses above, or becomes greater than, another value. A value (`Value1`) crosses above another value (`Value2`) whenever `Value1` is greater than `Value2` in the current bar but `Value1` was equal to or less than `Value2` in the previous bar.

`Above` is a synonym for `Over`.

Examples

```
   If Plot1 Crosses Above Plot2 Then
```

`Above` is used here to determine the direction of the cross of the values `Plot1` and `Plot2` on the bar under consideration.

```
   If Value1 Crosses Above Value2 Then
```

`Above` is used here to determine the direction of the cross of the values `Value1` and `Value2` on the bar under consideration.

AbsValue (Reserved Word)

Disclaimer

Returns the absolute value of num.

`AbsValue(num)`

Where (`num`) is a numeric expression.

Examples

`AbsValue(-25.543)` returns a value of 25.543.

`AbsValue(112.5)` returns a value of 112.5.

ActivityData (Reserved Word)

Disclaimer

This reserved word is a data alias used to reference the hidden data stream used by the ActivityBar study. When you want to refer to the ActivityBar data stream, and the reserved word that you are using is not an ActivityBar-related reserved word (thereby referencing the ActivityBar study data stream by default), you must use this data alias.

```
... of ActivityData
```

Notes

The reserved word `of` is used with `ActivityData` to make it easier to read.

Example

The following statement calculates the average of the last 10 closing prices of the ActivityBar study data stream. For instance, assume the ActivityBar study uses a data interval of 30 minutes and is applied to a daily chart. In this case, the statement calculates the average of the last ten 30-minute bars:

```
Value1 = Average(Close, 10) of ActivityData;
```

Additional Example

```
AB = AB_AddCellRange(High of ActivityData, Low of ActivityData, RightSide,
NumToStr(CellColor, 0), CellColor, 0);
```

`ActivityData` is used here to reference the High and Low values of the ActivityBar interval.

AddToMovieChain (Reserved Word)

Disclaimer

This reserved word adds .avi files to an existing video clip and returns a true/false value representing the success of the operation. If the reserved word was able to add the .avi file to the video clip, it returns a value of True; if it was not, it returns a value of False.

```
Print(AddToMovieChain(Movie_ID, Filename));
```

`Movie_ID` is a numeric expression representing the ID number of the video clip to which you're adding the .avi file, and `File` is a text string expression representing the full path and file name of the .avi file to add to the video clip.

Notes

When a video clip is played, it will play all the .avi files in the order they were added to the video clip.

Example

The following statements create a video clip and add two .avi files to it:

```
Variable: ID(-1);
ID = MakeNewMovieRef ;

Print(AddToMovieChain(ID, "c:\MyMovie.avi"));
Print(AddToMovieChain(ID, "c:\MyOtherMovie.avi"));
```

Ago (Reserved Word)

Disclaimer

Reserved word used to reference a specified number of bars back already analyzed by EasyLanguage.

Remarks

`Ago` can also be referred to as an offset, and can be represented by using notation consisting of a number in square brackets ([x]) where x is the number of points offset.

Example

```
Close of 1 Bar Ago
```

returns the Close price of the previous bar.

```
Average(Close, 10) of 1 Bar Ago
```

returns the Average of the last 10 Close prices as calculated on the previous bar.

Additional Example

```
Close[1]
```

is notation that also returns the Close price of the previous bar.

Alert (Reserved Word)

Disclaimer

This reserved word triggers an alert and enables you to provide a description of the condi tions that triggered the alert.

```
Alert("Description") ;
```

Description is any user-defined text string. You use the text string to provide information about the alert such as the market conditions that triggered it. This text string appears in the **Additional Info** field of the **Alert Notification** dialog box and in the Message Center. A description does not have to be provided, in which case the **Additional Info** field in the **Alert Notification** dialog box and the description for the alert entry in the Message Center are left blank.

Remarks

- A True value for Alert will only generate an alert when alerts have been enabled for the study in the Properties tab.
- Alerts are only generated for the last bar.

Example

If you include more than one Alert statement in your indicator or study, and more than one alert is triggered, the description included with the last alert triggered is the description shown. For example, assume the following indicator is applied to a price chart:

```
Plot1(Average(Close, 10), "Avg");
If Close Crosses Over Plot1 Then
Alert("Price crossed over average");
If Volume > Average(Volume,10) Then
Alert("Volume Alert");
```

If on the last bar of the price chart both conditions are true, both alerts are evaluated. In this case, only one alert is actually triggered and logged, and it will have the last description, which in the above example is the alert with the description "Volume Alert".

AlertEnabled (Reserved Word)

Disclaimer

This reserved word returns a value of True when the alert is enabled for the indicator or study applied to a price chart or grid (and False when it is not). This allows you to optimize the indicator or study for speed; the statements after this reserved word are evaluated only when the alert is enabled.

The difference between this reserved word and the `CheckAlert` reserved word is that `AlertEnabled` returns a value of True for all bars when the alert is enabled whereas `CheckAlert` returns a value of True only for the last bar on the chart.

Remarks

Alerts are only generated for the last bar.

Example

`AlertEnabled` will return True if alerts have been enabled.

Additional Example

the following statements calculate a cumulative advance/decline line and an alert is triggered when the cumulative advance/decline line hits a 50-bar high:

```
If AlertEnabled Then Begin
 If Close > Close[1] Then
  Value1 = Value1 + Volume
 Else
  Value1 = Value1 - Volume;
 If Value1 > Highest(Value1, 50)[1] Then
  Alert("New A/D line high");
End;
```

In this example, the advance/decline line will only be calculated if the alert is enabled, and it will be calculated for all bars on the price chart or in the grid, not just the last bar.

All (Reserved Word)

Disclaimer

Used in conjunction with "share(s)" or "contract(s)" in trading strategies specifying that all shares/contracts are to be sold (for long positions) or covered (for short positions) when exiting the current position.

Example

```
If Condition1 then
Sell all shares next bar at market;
```

An (Reserved Word)

Disclaimer

An is a Skip Word that performs no function within your code. It is used within a statement only to make the EasyLanguage code more readable.

Remarks

Skip words in EasyLanguage are not necessary, however, they sometimes make it easier to understand the purpose of the code.

Syntax Coloring in the EasyLanguage PowerEditor is dark green for all skip words by default.

Example

```
If an Close is > 100 Then
      Alert;
```

The word "an" is not necessary and the code functions the same way regardless of its presence.

And (Reserved Word)

Disclaimer

This boolean operator is used in condition statements to check multiple conditions.

```
AND
```

Remarks

And requires that in order for a condition to be True, all conditions must be True.

Examples

```
If Plot1 Crosses Above Plot2 AND Plot2 > 5 Then...
```

AND is used here to determine if the direction of the cross of the values Plot1 and Plot2, and that Plot2 is greater than 5 are both True on the bar under consideration. If either is False, the condition returns False.

```
If Value1 Crosses Above Value2 AND Value1 > Value1[1] Then...
```

AND is used here to determine if the direction of the cross of the variables Value1 and Value2, and that Value1 is greater that Value1 of one bar ago are both True on the bar under consideration. If either is False, the condition returns False.

Arctangent (Reserved Word)

Disclaimer

Returns the arctangent value (in degrees) of the specified (Num).

`ArcTangent(Num)`

Where `Num` is a numeric expression to be used in the calculation.

Remarks

The arctangent is the inverse of the tangent function.

`Num` should be the tangent of the desired the angle.

Examples

`ArcTangent(1)` returns a value of 45.

`ArcTangent(3.0776)` returns a value of `72`.

Array (Reserved Word)   same as Arrays

Disclaimer

Reserved word used to declare variable names that can contain multiple data elements.

Remarks

The declaration of an array creates a name that refers to a grouped set of data elements within an analysis techniques or strategy; these arrays can be numeric values, true/false comparisons, or text. Arrays allow you to group and reference data elements with an index number. Like variables you can organize and annotate your code with functional names that describe the nature of the calculation or purpose of the data.

Arrays in EasyLanguage start at data element 0, however, all built-in functions that operate on arrays ignore the zero element, so it is advised not to use the zero element.

An array may be described as a table of variables. That is, an array can hold many values just as a single variable holds a single value. Using an array has the advantage of allowing manipulation of the values in all or part of the array at once. That is, all the values can be averaged or sorted as a group. This would be a much more difficult task with individual variables. Additionally, an array is convenient for storing and retrieving values that occur on intermittent or non-consecutive bars.

The elements in an array may be organized in a single dimension or multiple dimensions. Once again, in spreadsheet terms, a 1-dimensional array has 1 column; a 2-dimensional array has multiple columns and rows; a 3-dimensional array has two 2-dimensional spreadsheets one on top of the other.

The Array reserve word has two forms: Arrays and Array, each is functionally equivalent and each must be followed by a colon : then a list of array names separated by commas (,) .

Note Array names like variables are unique to the study they are declared in; you can use the same name over again in any other study, but you cannot use the same name within the same study, for example, declaring a array with the same name as an variable. Also, remember to avoid naming arrays with the same name as an EasyLanguage reserved word.

Syntax-Common

```
Array: ArrayName(Value) ;
```

Where `ArrayName` is the unique name of the user-declared array, and `Value` is either a Numeric, True/False or String value used as the initial value of the array.

Syntax-Complete

Note that the angled brackets used in the declarations below denote that the parameter is optional. The angled brackets themselves are never typed into EasyLanguage code.

```
Array: <IntraBarPersist> <DataType> ArrayName(InitialValue<,datan>) <,
<IntraBarPersist> <DataType> VarName(InitialValue<,datan>)> ;
```

- `ArrayName` is the array declaration statement that may alternately be typed as `Arrays`.
- `IntraBarPersist` indicates that this array's values can be updated on every tick. By default, array values are only updated at the close of each bar.
- `DataType` is the optionally supplied data type (float, double, int, bool, string) of the array
- `Name` is the user specified array name that can be up to 20 characters long and must begin with an alphabetic character (names are not case sensitive)
- `InitialValue` is the initial value of the array elements
- `datan` is an optional parameter that identifies the data stream the array is tied to and may be data1 through data50.

Fixed Length Arrays

The declaration of a fixed length array must specify the maximum element reference number, and given an initial default value for each element; arrays are generally initialized to 0, but can be initial set to any useful value.

Examples

Usage Example: (Array Declaration, Single)

```
Array: int WeeklyHighs[52](0), int WeeklyLow[52](0)
```

(declares a single dimension 53 element array of integers, 0 to 52, and initializes each element to 0)

Usage Example: (Multidimensional Array Declaration)

```
Arrays: VolumeHighs[5,20](0), VolumeLow[5,20](0);
```

(declares a two dimensional array, 6 elements by 21 elements, or 126 elements, and initializes each element to 0)

Note Remember to avoid using data element 0 if you are going to use any of the built-in array functions.

Dynamic Arrays

The declaration of a dynamic array is the same as that of a fixed array except that the maximum element reference number is left blank. At this time, only single dimension dynamic arrays are supported.

Example

For example, to declare a dynamic array of integers named MyDynamicIntArray you would type the following into your EasyLanguage analysis technique:

```
Array: int MyDynamicIntArray[](0) ;
```

And to set the third element to 1 you would have to first increase the size of the array and then assign the value as follows:

```
Array_SetMaxIndex(MyDynamicIntArray, 10) ;
MyDynamicIntArray[2] = 1 ; //note that [2] is the third element since arrays
are zero-based
```

Other than having to resize the dynamic array prior to use dynamic arrays will behave exactly as existing arrays do. Note that while you can use a dynamic array in place of an existing array the reverse is not true. The new dynamic array functions will not accept static arrays as inputs. Note that this is due to the fact the dynamic and static arrays are treated differently internally.

Array_Compare (Reserved Word)

Disclaimer

This reserved function compares a range of elements between the between the first and second dynamic array and returns an integer value identifying any differences.

Usage

```
Array_Compare(SrcArrayName, SrcElement, DestArrayName, DestElement, NumElements);
```

Return

A value of 0 will be returned if all respective elements in the range of NumElements for both arrays are equal. If they are not, the following values will be returned:

1	A value in the range of SrcArrayName is greater than it's counterpart in DestArrayName.
-1	A value in the range of SrcArrayName is less than it's counterpart in DestArrayName.
-2	An error has occurred in the comparison (incompatible array types, Out of Bounds range error, etc.).

Parameters

SrcArrayName	A string that Identifies the first dynamic array to compare and is the name given in the array declaration
SrcElement	An integer representing the starting element to begin comparing in the first array.
DestArrayName	A string that Identifies the second dynamic array to compare with and is the name given in the array declaration.
DestElement	An integer representing the starting element to begin comparing in the second array.
NumElements	An integer representing the number of elements to compare.

Examples

Compares a range of 6 array elements starting with element 4 of the first dynamic array to element 5 of a second dynamic array.

```
Array_Compare(FirstArray, 4, SecondArray, 5, 6);
```

Array_Copy (Reserved Word)

Disclaimer

This reserved function copies NumElements elements from the dynamic array identified by SrcArrayName starting with the element at SrcElement and copies them to the array identified by DestArrayName starting with the element at DestElement. The SrcArrayName and DestArrayName can be the same value and in the case of an overlap this function will ensure that the original source elements are copied before being overwritten.

Usage

```
Array_Copy(SrcArrayName, SrcElement, DestArrayname, DestElement, NumElements);
```

Return

A value of 0 will be returned if the copy is successful, otherwise one of the following values will be returned

-1 – Unknown error

-2 – Invalid ScrArrayName of DestArrayName

-3 – Invalid SrcElement or DestElement

-4 – Destination array is different type than source array

-5 – Invalid or out of range NumElements

Parameters

`SrcArrayName`	A string that Identifies the dynamic array to copy from and is the name given in the array declaration
`SrcElement`	An integer representing the starting element to begin copying from in the source array.
`DestArrayName`	A string that Identifies the dynamic array to copy to and is the name given in the array declaration
`DestElement`	An integer representing the starting element to begin copying to in the destination array.
`NumElements`	An integer representing the number of elements to copy.

Examples

Copies a range of 12 array elements from element 5 of the first dynamic array to element 8 of a second dynamic array.

```
Array_Copy(FirstArray, 5, SecondArray, 8, 12);
```

Array_GetMaxIndex (Reserved Word)

Disclaimer

Returns the index of the last element of a dynamic array.   Note that the actual number of elements that an array can hold is the MaxIndex+1.

Usage

```
Array_GetMaxIndex(ArrayName);
```

Return

An integer value greater than or equal to 0 will be returned if the function is successful. This value is equivalent to the MaxIndex value used when declaring a static array.

Parameters

`ArrayName`	Identifies the dynamic array for which you want to get the size and is the name given in the array declaration

Examples

Value 2 will contain the maximum index value of 8 for the specified dynamic array.

```
Array: MyDynArray[](0);
Condition1 = Array_SetMaxIndex(MyDynArray,8);
... other easylanguage statements

Value2 = Array_GetMaxIndex(MyDynArray);
```

Array_GetType (Reserved Word)

Disclaimer

Identifies the data type of the elements in the dynamic array identified by ArrayName

Usage

```
Array_GetType(ArrayName);
```

Return

A value of -1 will be returned if ArrayName is invalid, otherwise the function will return one of the following values:

1 – Unknown

2 – Boolean

3 – String

4 – Integer

6 - Float

7 - Double

Parameters

`ArrayName`	Identifies the dynamic array to inspect and is the name given in the array declaration.

Examples

Value2 will a type of 4 (integer) for the specified dynamic array.

```
Array: int MyDynArray[](0);
Condition1 = Array_SetMaxIndex(MyDynArray,8);
... other easylanguage statements

Value2 = Array_GetType(MyDynArray);
```

Array_SetMaxIndex (Reserved Word)

Disclaimer

Resizes a dynamic array. A dynamic array can be made larger or smaller and if made larger each new element will be initialized to the value used when the dynamic array was first created.

Usage

```
Array_SetMaxIndex(ArrayName, MaxIndex);
```

Parameters

`ArrayName`	Identifies the dynamic array to resize and is the name given in the array declaration
`MaxIndex`	An integer representing the index of the last element and is equivalent to the MaxIndex value used when declaring a static array. This value can be greater than or equal to 0.

Remarks

Returns a boolean value indicating success or failure.

Examples

Changes the size of a newly created dynamic array to a maximum index value of 7.

```
Array: MyDynArray[](0);
... other easylanguage statements
Condition1 = Array_SetMaxIndex(MyDynArray, 7);
```

Array_SetValRange (Reserved Word)

Disclaimer

This reserved function places the value Val into each element of the dynamic array identified by ArrayName starting with element BegElementNum and ending with element EndElementNum.

Usage

```
Array_SetValRange(ArrayName, BegElementNum, EndElementNum, Val);
```

Return

A value of 0 will be returned if the function is successful in setting the values, otherwise one of the following values will be returned:

-1 – Unknown error

-2 – Invalid ArrayName

-3 – BegElementNum or EndElementNum is out of range

-4 – Val is wrong type

Parameters

`ArrayName`	Identifies the source dynamic array to compare and is the name given in the array declaration
`BegElementNum`	An integer representing the first element in the range of values to be set to the value Val
`EndElementNum`	An integer representing the last element in the range of values to be set to the value Val
`Val`	Identifies the value to be set and can be a boolean, string, integer, float or double.

Examples

Sets elements 5 through 12 of the specified array to a string value of "OK".

```
Value1 = Array_SetValRange(MyDynArray, 5, 12, "OK");
```

Array_Sort (Reserved Word)

Disclaimer

This reserved function sorts a range of elements in a dynamic array in either ascending or descending order..

Usage

```
Array_Sort(ArrayName, BegElementNum, EndElementNum, SortOrder);
```

Return

An integer value of 0 will be returned if the function is successful in setting the value else one of the following values will be returned:

-1 – Unknown error

-2 – Invalid ArrayName

-3 – BegElementNum or EndElementNum is out of range

-4 – SortOrder is invalid

Parameters

`ArrayName`	Identifies the source dynamic array containing elements to be sorted and is the name given in the array declaration
`BegElementNum`	An integer representing the first element in the range of sorted values.
`EndElementNum`	An integer representing the last element in the range of sorted values.
`SortOrder`	An True/False value that specifies if the sort is Ascending(True) or Descending(False) .

Examples

Performs an ascending sort on the specified dynamic array from elements 1 through 5.

```
Value1 = Array_Sort(MyDynArray, 1, 5, True);
```

Array_Sum (Reserved Word)

Disclaimer

This reserved function that sums a range of elements in the specified dynamic array.

Usage

```
Array_Sum(ArrayName, BegElementNum, EndElementNum);
```

Return

Returns the numeric sum of the values in the specified range If the array contains string or boolean (true/false) elements, a 0 is returned.

Returns the numeric sum of the values in the specified range If the array contains string elements, a 0 is returned. If the array contains Boolean (True/False) elements, the number of True elements are returned.

Parameters

`ArrayName`	Identifies the source dynamic array containing elements to be summed and is the name given in the array declaration
`BegElementNum`	An integer representing the first element in the range of values to be summed
`EndElementNum`	An integer representing the last element in the range of values to be summed

Examples

Value1 will contain the sum of values contained in the specified dynamic array from elements 4 through 10.

```
Value1 = Array_Sum(MyDynArray, 4, 10);
```

ARRAYSIZE (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage.

ARRAYSTARTADDR (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

ASK (Reserved Word)

Disclaimer

Ask is an OptionStation reserved word, referring to the modeled ask value of an option.

AskDate (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the EasyLanguage date of the last ask transmitted by the datafeed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

AskDateEX (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the Julian date (accurate to the second) of the last ask transmitted by the datafeed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

AskExch (Reserved Word)

Disclaimer

A quote field that returns a text string representing the exchange the last ask was sent from.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

AskSize (Reserved Word)

Disclaimer

Returns a numeric expression representing the number of units offered at the best ask price.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

AskTime (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the time of the last ask price.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

AskTimeEX (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the time (accurate to the second) of the last ask price.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

ASSET (Reserved Word)

Disclaimer

ASSET is an OptionStation reserved word, It is a modifier to other Asset related reserve words.

Example

Close of Asset;

ASSETTYPE (Reserved Word)

Disclaimer

AssetType is an OptionStation reserved word that identifies the underlying asset category (stock, index, future) of the OptionStation window.

ASSETVOLATILITY (Reserved Word)

Disclaimer

ASSETVOLATILITY is an OptionStation reserved word, reserved for future use.

At (Reserved Word)

Disclaimer

AT is a Skip Word that performs no function within your code. It is used within a statement only to make the EasyLanguage code more readable.

Remarks

Skip words in EasyLanguage are not necessary, however, they sometimes make it easier to understand the purpose of the code.

Syntax Coloring in the EasyLanguage PowerEditor is dark green for all skip words by default.

Example

```
If Close > 100 Then
      Buy next bar at Market;
```

The word "at" is not necessary and the code functions the same way regardless of its presence.

At$ (Reserved Word)

Disclaimer

Used in trading strategies to anchor exit prices to the bar where the entry order was placed. At$ must be used in conjunction with an entry order that has an assigned label.

Example

The following order buys when the 10-bar moving average crosses over the 20-bar moving average, placing a stop loss order 1 point under the Low of the bar where the cross over occurred by appending At$ to the Sell statement.

```
If Average(Close, 10) Crosses Over Average(Close, 20) then
 Buy ("MA Cross Over") next bar at market;
Sell next bar from entry ("MA Cross Over") At$ Low - 1 point stop;
```

AtCommentaryBar (Reserved Word)

Disclaimer

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 EasyLanguage to skip all commentary-related calculations for all bars except for the one where the commentary is requested.

```
AtCommentaryBar
```

The difference between `AtCommentaryBar` and `CommentaryEnabled` is that `CommentaryEnabled` returns a value of True for ALL bars when the Analysis 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 Analysis 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));
```

Additional Example

More than one statement can be executed on the selected bar by using nested statement syntax:

```
If AtCommentaryBar Then Begin
 {Your Commentary Code Here}
End;
```

AUTOSESSION (Reserved Word)

Disclaimer

AUTOSESSION is used with session information to refer to auto-detect sessions.

AvgBarsEvenTrade (Reserved Word)

Disclaimer

Returns the Average number of bars that elapsed during even trades for all closed trades.

```
AvgBarsEvenTrade
```

Examples

`AvgBarsEvenTrade` returns 3.00 if the number of bars elapsed during 4 even trades were 4, 3, 6, and 2.

`AvgBarsEvenTrade` returns 5.00 if the number of bars elapsed during 2 even trades were 7, and 3.

AvgBarsLosTrade (Reserved Word)

Disclaimer

Returns the Average number of bars that elapsed during losing trades for all closed trades.

```
AvgBarsLosTrade
```

Examples

`AvgBarsLosTrade` returns 3.00 if the number of bars elapsed during 4 losing trades were 4, 3, 6, and 2.

`AvgBarsLosTrade` returns 5.00 if the number of bars elapsed during 2 losing trades were 7 and 3.

AvgBarsWinTrade (Reserved Word)

Disclaimer

Returns the Average number of bars that elapsed during winning trades for all closed trades.

`AvgBarsWinTrade`

Examples

`AvgBarsWinTrade` returns 3.00 if the number of bars elapsed during 4 winning trades were 4, 3, 6, and 2.

`AvgBarsWinTrade` returns 5.00 if the number of bars elapsed during 2 winning trades were 7, and 3.

AvgEntryPrice (Reserved Word)

Disclaimer

Returns the Average entry price of each open entry in a pyramided position.

```
AvgEntryPrice
```

Remarks

`AvgEntryPrice` only returns the average entry price for open trades.

Examples

`AvgEntryPrice` returns 150 if three trades are currently open and were entered at a price of 130, 145 and 175.

`AvgEntryPrice` returns 50 if four trades are currently open and were entered at a price of 42, 53, 37 and 68.

AvgList (Reserved Word)

Disclaimer

Returns the average value of the inputs.

```
AvgList(Num1, Num2, Num3, etc.)
```

Where `Num1` is a numeric expression representing a value to be used in the calculation.

`Num2` is a second numeric expression representing a value to be used in the calculation.

`Num3` is a third numeric expression representing a value to be used in the calculation.

Remarks

The `AvgList` calculates a simple average of all of the values of (Numx). The sum of all values of Num divided by the number of members.

Examples

`AvgList(45, 72, 86, 125, 47)` returns a value of 75.

`AvgList(18, 67, 98, 24, 65, 19)` returns a value of 48.5.

Bar (Reserved Word)

Disclaimer

References values for a specific bar based on the data interval.

Remarks

Bar is normally used with the reserved word Ago to specify a value or condition (usually 1) "Bar Ago". When the data interval is set to Daily, this and the reserved word Day are identical.

Examples

```
Close of 1 Bar Ago
```

returns the Close price of the previous bar.

```
BuyToCover this bar on close
```

orders a trading strategy to exit all short positions on the close of the current bar.

Additional Example

```
Buy ("Order Name") next bar at open
```

will buy during the formation of the next bar if a price meets the open price of the bar.

BarInterval (Reserved Word)

Disclaimer

Reserved word that returns the bar interval for the type of data that an analysis technique is applied to.

Remarks

`BarInterval` is only valid when used on intraday, volume, and tick data. Returns minutes for intraday time-based bars, shares/contracts for volume-based bars, or tick count for tick-based bars.

Examples

```
Condition1 = (BarInterval = 5)
```

is a statement that will cause `Condition1` to be true if the analysis technique is applied to a 5-minute intraday chart..

```
CalcTime(Sess1StartTime, BarInterval)
```

will add the bar interval to the start time of the asset in an intraday time-based chart.

Additional Example

`CalcTime(Sess1EndTime, -BarInterval)` returns the time of the last bar before the close of the trading session for an intraday time-based chart.

Bars (Reserved Word)

Disclaimer

References values for a specific bar based on the data interval.

Remarks

The reserved word Bars is normally used with the reserved word Ago to specify a value or condition (usually 1) "Bars Ago". When the data interval is set to Daily, this and the reserved word Days are identical.

Examples

```
Close of 5 Bars Ago
```

returns the Close price of the bar 5 bars previous to the current bar.

BarsSinceEntry (Reserved Word)

Disclaimer

Returns the number of bars since a specified position was entered.

```
BarsSinceEntry(Num)
```

Where `Num` is a whole number (up to a maximum 10) representing positions ago.

Remarks

This function can only be used in the evaluation of strategies.

Remarks

This function can only be used in the evaluation of strategies. If no input is provided, the current position (0) is assumed.

Example

```
Value1 = BarsSinceEntry(2);
```

In this example, `Value1` might return a value of 68 if it has been 68 bars since the entry that occurred 2 positions ago.

BarsSinceExit (Reserved Word)

Disclaimer

Returns the number of bars ago since a specified position was closed out.

```
BarsSinceExit(Num)
```

Where `Num` is a whole number (from 1 of 10) representing positions ago.

Remarks

This function can only be used in the evaluation of strategies.

Example

```
Value1 = BarsSinceExit(2);
```

In this example, `Value1` might return a value of 46 if it has been 46 bars since the exit that occurred 2 positions ago.

BarStatus (Reserved Word)

Disclaimer

This reserved word is used with multiple data series (same symbol, different data intervals) or ActivityBar studies to determine whether the current tick is the opening or closing tick of a bar in the other data series, or whether it is a trade ‘inside the bar.’

```
BarStatus(DataNum)
```

`DataNum` is a numeric expression representing the data stream that is being evaluated. 1 refers to Data1, 2 to Data2 and so on. `DataNum` can be between 1 and 50, inclusive.

Notes:

This reserved word will return one of four possible values:

2 = the closing tick of a bar
1 = a tick within a bar
0 = the opening tick of a bar (relevant only for strategies using Open Next Bar order actions)
-1 = an error occurred while executing the reserved word

The data series referenced must be applied to the chart. For example, to use `BarStatus` (2), a second data series must be applied to the chart.

Example

To perform an operation when the ActivityBar is the last trade of the ‘big’ bar, you could write:

```
If BarStatus(1) = 2 Then
 {Your Operation Here} ;
```

… where Data1 is the price chart, and your analysis technique is an ActivityBar study.

Additional Example

The following statements reset the numeric variable `Value1` to 0 when the bar to which the ActivityBar study is applied is closed:

```
If BarStatus(1) = 2 Then
 Value1 = 0
Else
 Value1 = Value1 + 1;
```

BarType (Reserved Word)

Disclaimer

Returns a number indicating the interval setting of the price data an analysis technique is applied to.

Remarks

The number returned is based on the data interval of the price data. BarType will return:

0 = TickBar
1 = Intraday
2 = Daily
3 = Weekly
4 = Monthly
5 = Point & Figure

Examples

BarType returns 0 when applied to price data based on tick interval.

BarType returns 2 when applied to price data based on daily interval.

Additional Example

To assure that a statement is executed only on a daily chart we can write:

```
If BarType = 2 then Sell next bar at market;
```

Based (Reserved Word)

Disclaimer

This word is a skip word retained for backward compatibility.

Remarks

This word is skipped in EasyLanguage and is not necessary. These skipped words however, do make it easier to understand the purpose of the code.

By default, unnecessary words and sections marked as comments will appear dark green in the PowerEditor.

Begin (Reserved Word)

Disclaimer

This word is used to begin a series of code that should be executed on the basis of an `If… Then`, `If… Then… Else`, `For`, or `While` statement.

```
Begin
```

Remarks

`Begin` is only necessary if using more than one line of code after an `If… Then`, `If… Then… Else`, `For`, or `While` statement.

Each `Begin` must have a corresponding `End`.

Examples

```
If Condition1 Then Begin
  {Your Code Line1}
  {Your Code Line2, etc.}
End;
```

`Begin` is used here to include the execution of Line1 and Line2 only when `Condition1` is `True`.

```
If Condition1 Then Begin
  {Your Code Line1}
  {Your Code Line2, etc.}
End
Else Begin
  {Your Code Line3}
  {Your Code Line4, etc.}
End;
```

`Begin` is used here twice to include the execution of Line1 and Line2 only when `Condition1` is `True` and execute Line3 and Line4 only when `Condition1` is `False`.

Below (Reserved Word)

Disclaimer

This word is used to check for the direction of a cross between values.

```
Below
```

Remarks

Used in conjunction with the reserved word "`Crosses`" to detect when a value crosses below, or becomes less than another value. A value (`Value1`) crosses below another value (`Value2`) whenever `Value1` is less than `Value2` in the current bar but `Value1` was equal to or greater than `Value2` in the previous bar.

`Below` is a synonym for `Under`.

Examples

```
If Plot1 Crosses Below Plot2 Then ..
```

`Below` is used here to determine the direction of the cross of the values `Plot1` and `Plot2`.

```
If Value1 Crosses Below Value2 Then ..
```

`Below` is used here to determine the direction of the cross of the variables `Value1` and `Value2`.

Beta (Reserved Word)

Disclaimer

A quote field that returns the most recent Beta value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

Beta_Down (Reserved Word)

Disclaimer

Retained for backward compatibility.

Beta_Up (Reserved Word)

Disclaimer

Retained for backward compatibility.

BID (Reserved Word)

Disclaimer

Bid is an OptionStation reserved word, referring to the modeled bid value of an option.

BidDate (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the EasyLanguage date of the last bid transmitted by the datafeed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

BidDateEX (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the Julian date (accurate to the second) of the last bid transmitted by the datafeed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

BidDirectionNNM (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

BidExch (Reserved Word)

Disclaimer

A quote field that returns a string expression representing the exchange the last bid was sent from.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

BidSize (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the number of units bid at the best bid price.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

BidTime (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the time of the last bid price.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

BidTimeEX (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the time (accurate to the second) of the last bid price.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

BigPointValue (Reserved Word)

Disclaimer

Returns a numeric expression representing the market value of a whole number price move for the share or contract price of a particular symbol. The BigPointValue value is represented by a whole number. For example; the BigPointValue of the E-Mini S&P is 50, which represents the dollar value of a full one point move in the E-Mini S&P. Any stock would have a BigPointValue of 1 since a full point move in a share price represents a change in market value of 1.

```
BigPointValue
```

Examples

`BigPointValue` returns the market value of a full point move in the price of a share or contract.

`MinMove / PriceScale * BigPointValue` returns the market value of the minimum price move for a share or contract..

See Also

PriceScale

PointValue

MinMove

Black (Reserved Word)

Disclaimer

Sets the plot color or background color to Black.

```
Black
```

Remarks

There are 16 color names available in EasyLanguage. For more information, see EasyLanguage Colors and Their Corresponding Numeric Values.

Examples

```
Plot1(Value1, "Test", Black)
```

plots `Value1` with the name `Test`, and sets the color of `Plot1` to `Black`.

```
TL_SetColor(1, Black)
```

sets the color of a TrendLine with a reference number of 1 to `Black`.

BlockNumber (Reserved Word)

Disclaimer

Returns the Block Number of the security block attached to the machine.

Remarks

The `BlockNumber` function can be used to check for the presence of a particular security block before plotting an indicator.

Examples

If you wanted to only display an indicator when the user had block number 55522 attached to the machine, you could use the following syntax:

```
If BlockNumber = 55522 Then
 Plot1(Value1, "Indicator");
```

Blue (Reserved Word)

Disclaimer

Sets the plot color or background color to Blue.

```
Blue
```

Remarks

There are 16 colors available in EasyLanguage. For more information, see EasyLanguage Colors and Their Corresponding Numeric Values.

Examples

```
Plot1(Value1, "Test", Blue)
```

plots `Value1` with the name `Test`, and sets the color of `Plot1` to `Blue`.

```
TL_SetColor(1, Blue)
```

sets the color of a TrendLine with a reference number of 1 to `Blue`.

Book_Val_Per_Share (Reserved Word)

Disclaimer

Retained for backward compatibility.

BOOL (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

BoxSize (Reserved Word)

Disclaimer

Reserved word used to refer to the minimum change in price that is required for the addition of an X to the top or an O to the bottom of a Point & Figure column.

Remarks

The `BoxSize` is set when creating a Point & Figure chart.

Break (Reserved Word)

Disclaimer

The Break reserved word terminates the execution of the nearest enclosing loop or conditional statement in which it appears. Control passes to the statement that follows the terminated statement, if any.

Syntax

```
Break;
```

Remarks

When used in a conditional Switch statement, Break causes EasyLanguage to execute the next statement after the switch 'end'.

In loops (for-loop, while-loop, or repeat/until), Break terminates execution of the loop containing the Break statement, with control passing to the next EasyLanguage statement after the loop. Within nested statements, Break terminates only the loop that immediately encloses it.

Example

In the following example, the For loop will exit after printing values of 1 through 4.

```
For i = 0 to 9
  Begin
    Print(i);
    If (i=4) then Break;
  End;
```

BreakEvenStopFloor (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word SetBreakEven.

BreakPoint (Reserved Word)

Disclaimer

Used with the EasyLanguage Debugger, when placed in your EasyLanguage code it automatically opens the EasyLanguage Debugger window when encountered as the study calculates.

`BreakPoint` is a compiler directive that opens the EasyLanguage Debugger when encountered.

See About EasyLanguage Debugger for more information

Usage

```
BreakPoint ("BreakPointNameID");
```

Parameters

BreakPointNameID	A string expression or string variable. This is a required parameter.

Remarks

Each occurrence of `BreakPoint` must be named. The name parameter may specified as a string expression, following the keyword, using the form: `BreakPoint` ("MyBreakName"). A string variable can also be used as the name or in combination with a literal string value such as: BreakPoint("Break ID" + GetSymbolName).

Important While the EasyLanguage Debugger is open, activity in all TradeStation windows is halted. No data, alerts, or strategy orders will be generated. Any window, including Chart Analysis, in open workspaces will not update until the EasyLanguage Debugger is closed. The EasyLanguage Debugger is intended for use in a development environment. Debugging should not take place while trading or using strategy automation.

Examples

Open the EasyLanguage Debugger from within a loop.

```
For Counter = 0 to Length -1 Begin
    Sum = Sum + Close[Counter];
    BreakPoint("BreakID");
End;
```

Buy (Reserved Word)

Disclaimer

The Buy order is used to establish a long position. If there is an existing open short position, an order to cover the short position is sent first. Once the short position is closed, the Buy order is sent to establish the long position. The order specifications are defined by parameters in the Buy statement.

```
Buy [("Order Name")] [Number of Shares/Contracts] [Order Action];
```

Only the word `Buy` and the `Order Action` are required by the order syntax rules. For example:

```
Buy next bar at open;
```

or

```
Buy next bar at 45 stop;
```

If no order name is specified, the default name for the order will be `"Buy"`. If the number of shares (or contracts) is not specified, then the number of shares is determined by the amount specified in the **Trade size** section of the **Strategy Properties for all Strategies** dialog box.

Each portion of the statement, `Order Name`, `Number of Shares`, and `Order Action` are described next.

Order Name

When using multiple long entries within a strategy, it is helpful to label each entry order with a different name. By naming entry orders, you can easily identify all positions, both on the chart and in the Strategy Performance Report under the **Trades** tab. Also, naming the entry orders allows you to tie an exit to a particular entry order. For more details, see Sell.

To name a long entry order, include a descriptive name in quotation marks and within parenthesis after `Buy`. For example:

```
Buy ("MyFirstEntry") next bar at open;
```

This instruction initiates a long position named `"MyFirstEntry"`.

Number of Shares/Contracts

To specify how many shares (or contracts) to buy, place a numeric expression followed by the words `shares` (or `contracts`) after `Buy` and after the entry order name, if used. For example:

```
Buy ("My Entry") 100 shares next bar at open;
```

```
Buy 5 contracts next bar at 100.50 limit;
```

`Buy Value1 shares next bar at 100.50 limit;` (Where `Value1` is a variable value calculated within the strategy.)

Note The words `shares` and `contracts` are synonymous.

If the number of shares/contracts is not specified, the value entered under the **Trade size** section of the **Strategy Properties for all Strategies** dialog box is used. The **Trade size** section controls the number of contracts or shares traded; this can be set to either a fixed number of shares/contracts or dollars per transaction. The trading strategy engine uses whatever is specified in the strategy code and will override the **Strategy Properties for all Strategies** settings.

Order Action

You must specify one of the five different order actions with your `Buy` order.

```
... this bar on close;
```

```
... next bar at open;
```

```
... next bar at market;
```

```
... next bar at yourprice Stop;
```

```
... next bar at yourprice Limit;
```

Strategy orders are generated on the bar close event and placed in the market based on the order action. For example:

```
Buy this bar on close;
```

When this order is generated, the following actions are taken:

- A Buy arrow is placed on the current bar at the close tick mark.
- If the order is generated on an intra-day bar and not the last bar on the chart, the order is sent directly into the market as a market order.

- If the order is generated on an intra-day bar and it is the last bar on the chart, or it is a daily, weekly, or monthly chart, then the order is sent directly to the after hours market as a Day+ limit order at the price at the close of the bar. If it is not filled in the after hours session, then the order will be placed as a limit order on the open of the next trading day.

For example:

```
Buy next bar at market;
```

When this order is generated, the following actions are taken:

- A Buy arrow is placed on the next bar at the open tick mark.
- If the order is generated on an intra-day bar and not the last bar on the chart, the order is sent directly to the market as a market order.
- If the order is generated on an intra-day bar and it is the last bar on the chart, or it is a daily, weekly, or monthly chart, then the order is held until the open of the next trading day and sent as a market order.

For example:

```
Buy next bar at 50 limit;
```

When this order is generated, the following actions are taken:

- If the stock or future trades at 50 or less, a buy arrow is placed on the next bar at the first tick price of 50 or less.
- If the order is generated on an intra-day bar and not the last bar on the chart, the order is sent directly to the market as a limit order.
- If the order in generated on an intra-day bar and it is the last bar on the chart, or it is a daily, weekly, or monthly chart, then the order is held until the open of the next trading day and sent as a limit order.

For example:

```
Buy next bar at 50 stop;
```

When this order is generated, the following actions are taken:

- If the stock or future trades at 50 or higher, a buy arrow is placed on the next bar at the first tick price of 50 or higher.
- If the order is generated on an intra-day bar and not the last bar on the chart, the order is held in the TradeManager window until the stop price is penetrated; then an order is sent directly to the market as a market order.
- If the order is generated on an intra-day bar and it is the last bar on the chart, or it is a daily, weekly, or monthly chart, then the order is held until the open of the next trading day and held in the TradeManager window until the stop price is penetrated; then an order is sent directly to the market as a market order.

Stop orders are held on your computer and are not reflected in the market until the stop price is penetrated. If you turn off your computer, the order will not have an opportunity to be placed and filled.

Additional Examples

This statement buys the default number of contracts/shares specified in the **Trade size** section of the **Strategy Properties for all Strategies** dialog box at the open of the next bar, and names this entry order Entry#1:

```
Buy ("Entry#1") next bar at market;
```

The next statement places an order to buy 5 contracts at the high of the current bar plus the range of the current bar, or any price higher. Note that Range is an EasyLanguage function that returns the high minus the low. This order remains active throughout the next bar (until filled or canceled):

```
Buy 5 contracts next bar at High + Range stop;
```

The next statement places an order to buy 100 shares at the lowest low of the last 10 bars, or any price lower. This order remains active throughout the next bar (until filled or canceled), and the order is named LowBuy:

```
Buy ("LowBuy") 100 shares next bar at Lowest(Low, 10) limit ;
```

BuyToCover (Reserved Word)

Disclaimer

This trading verb is used to cover a short position. The specifics of the order are defined by the optional components used in the statement (that is, how many shares/contracts, at what price, and so on).

Exit orders do not pyramid. Once the exit criteria are met and the exit order filled, the order is ignored for that position until the position is modified (that is, more shares/ contracts are sold or a new short position is established).

```
BuyToCover [("Order Name")] [from entry ("Entry Name")] [ Number of Shares/Contracts
    [Total]] [Order Action];
```

Only the word `BuyToCover` and the `Order Action` are required to exit a short position. The following is a complete Easy Language statement:

```
BuyToCover This Bar on Close;
```

If no other parameters are specified, the default value for the `Order Name` is `"Cover"`, and all short contracts will be exited from the position.

Each portion of the statement, `Order Name`, `Number of Shares`, `from entry` and `Order Action` are described next.

Order Name

When using multiple exits within a strategy, it is helpful to label each exit order with a different name. This enables you to identify these exit orders in both the price chart and the Strategy Performance Report. To assign an exit order a name, specify a name in quotation marks and within parentheses immediately after the word `BuyToCover`. For Example:

```
BuyToCover ("My Exit") This Bar on Close;
```

This instruction closes the entire short position, and the order is labeled `My Exit`.

Tying an Exit to an Entry

It is possible to tie an exit instruction to a specific entry. This can be achieved only if you named the short entry, and the short entry is in the same strategy as the exit. Consider the following strategy:

```
SellShort ("MyShort") 10 Shares Next Bar at Market;
SellShort 20 Shares Next Bar at Low - 1 Point Stop ;

BuyToCover From Entry ("MyShort") Next Bar at Low - 3 Points Stop;
```

In the above example, the strategy may short 30 shares total; your short position is 30 shares. However, the `BuyToCover` instruction only closes out the 10 shares shorted using the `MyShort` entry order. It ignores any other order, and does not close out the other 20 shares. Therefore, this strategy leaves you short 20 shares.

You can also close part of an entry order. For example, if your entry, which you named "`MyShort`" buys 10 shares, you can specify that you want to exit from entry "`MyShort`" but only close out 5 shares, not the entire 10:

```
BuyToCover From Entry ("MyShort") 5 Shares Next Bar at Low - 3 Points Stop;
```

Important The entry name is case sensitive. Be sure to use consistent capitalization. Also, it is important to remember that exit orders do not pyramid; therefore, if an exit does not close out a position, you will need another exit order (or reversal order) in order to close out a position.

Number of Shares/Contracts

To specify how many shares/contracts to close out, use a numeric expression followed by the word `shares` or `contracts` after the trading verb `BuyToCover`. Some Examples:

```
BuyToCover 100 Shares This Bar on Close;

BuyToCover From Entry ("MovAvg") 10 Shares Next Bar at Low - 1 Point Stop;
```

Note The words `shares` and `contracts` are synonymous.

If you do not specify the number of shares/contracts in the `BuyToCover` instruction, the exit order closes out the entire short position, rendering your position flat.

When you specify the number of shares/contracts, the `BuyToCover` instruction exits the specified number of shares/contracts from every open entry.

Therefore, if the Strategy allows for pyramiding, and has shorted 500 shares three times (for a total of 1,500 shares), and an order to `BuyToCover 100 Shares` is placed by the Strategy, the exit order will exit a total of 300 shares: 100 shares from each one of the three entries.

However, if you want to exit a total of 100 shares, you can use the reserved word `Total` in the `BuyToCover` instruction. Using the word `Total` causes the Strategy to exit 100 shares from the first open entry (first in, first out). For Examples:

```
BuyToCover 100 Shares Total This Bar on Close;
BuyToCover From Entry ("MovAvg") 10 Shares Total Next Bar at Low - 1 Point Stop ;
```

Order Action

You must specify one of the five different order actions with your BuyToCover order.

```
... this bar on close;
... next bar at open;
... next bar at market;
... next bar at yourprice Stop;
... next bar at yourprice Limit;
```

The order action this bar on close is provided for backtesting purposes only; it enables you to back test 'market at close' orders, which you cannot automate using TradeStation. Given that all orders are evaluated and executed at the end of each bar, TradeStation reads and issues the this bar on close order once the bar has closed (for example, once the daily trading session has ended). TradeStation fills the order using the close of the current bar, but you have to place an order at market to be executed on the next bar. This invariably introduces slippage.

The order action BuyToCover next bar at *price* Limit instructs TradeStation to exit a short position at the first opportunity at the specified *price* or lower. The order action BuyToCover next bar at *price* Stop instructs TradeStation to exit at the first opportunity at the specified price or higher.

Stop and limit orders are treated by TradeStation as market if touched (MIT) orders. A MIT order becomes a market order when the price of the symbol meets the specified order price. It is possible for stop and limit orders not to be filled (that is, price never met); in this case, the orders are canceled at the close of the bar.

Tying the Exit Price to the Bar of Entry

When specifying the execution method, you can vary stop and limit orders by using 'At$' instead of 'at'. Using At$ forces the strategy to refer to the value the numerical expression *price* had on the bar where the entry order was generated. Consider the following statement:

```
BuyToCover From Entry ("MySell") Next Bar At$ High + 1 Point Stop;
```

The above statement places an order to exit the short position at one point higher than the high of the bar where the order to establish the short position was generated (for example, if an order to SellShort next bar... is generated today, the prices referenced will be today's, not tomorrow's. Even though the order was placed and filled tomorrow, it was generated today and that is the bar referenced).

To use the At$ reserved word, you must name the entry order, and the BuyToCover instruction must refer to the specific entry order.

As another example, if the maximum risk you will tolerate for a position is 5 points over the closing price of the bar on which you generated the entry order, you can use the following statement:

```
BuyToCover From Entry ("MySell") Next Bar At$ Close + 5 Points Stop;
```

This is a valuable technique that allows you to refer easily to the prices of the bar on which the entry order was generated.

Example

This statement exits all contracts/shares of your open short position at the close of the current bar. Your position will be flat.

```
BuyToCover This Bar on Close;
```

Additional Example

The next instruction exits all contracts/shares of your positions opened by the entry order Entry#1 at the open of the next bar, and this order is named ShortExit.

```
BuyToCover ("ShortExit") From Entry ("Entry#1") Next Bar at Market;
```

The following statement places an order to close 5 contracts in total at the high of the current bar plus 1 point or anything higher. This order is active throughout the next bar (until filled or canceled):

```
BuyToCover 5 Contracts Total Next Bar at High + 1 Point Stop;
```

The next instruction places an order to exit 100 shares out of every entry at the low minus the range of the current bar or anything lower. This order is active throughout the next bar (until filled or canceled), and is named LowExit.

```
BuyToCover ("LowExit") 100 Shares Next Bar at Low - Range Limit;
```

The following statement enables you to monitor your risk by placing an exit order 5 points over the closing price of the bar that generated the short entry order:

```
BuyToCover From Entry ("MySell") Next Bar At$ Close + 5 Points Stop;
```

By (Reserved Word)

Disclaimer

By is a Skip Word that performs no function within your code. It is used within a statement only to make the EasyLanguage code more readable.

Remarks

Skip words in EasyLanguage are not necessary, however, they sometimes make it easier to understand the purpose of the code.

Syntax Coloring in the EasyLanguage PowerEditor is dark green for all skip words by default.

BYTE (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

C (Reserved Word)

Disclaimer

Shortcut notation that returns the Close of the bar.

Remarks

Anytime the Close of a bar is needed, the letter `C` can be used in an equivalent fashion.

Examples

```
C of 1 bar ago
```

returns the Close price of the previous bar.

```
Average(C, 10)
```

returns the Average of the last 10 Close prices.

Additional Example

To check that the last two bars have Close prices higher than the previous bar, the following language can be used in a ShowMe study:

```
If C > C[1] and C[1] > C[2] then
 Plot1(High, "ClosedUp");
```

CALL (Reserved Word)

Disclaimer

CALL is a constant reserve word returning the value 3.

CALLCOUNT (Reserved Word)

Disclaimer

CALLCOUNT is an OptionStation reserved word, referring to the number of calls in the OptionStation option(n) array.

CALLITMCOUNT (Reserved Word)

Disclaimer

CALLITMCOUNT is an OptionStation reserved word, reserved for future use.

CallOpenInt (Reserved Word)

Disclaimer

Returns the daily total Call option open interest of all the options for an underlying stock, index, or future. Allows historical bars back reference.

Example

```
Value1 = CallOpenInt;
```

CALLOTMCOUNT (Reserved Word)

Disclaimer

CALLITMCOUNT is an OptionStation reserved word, reserved for future use.

CALLSERIESCOUNT (Reserved Word)

Disclaimer

CALLSERIESCOUNT is an OptionStation reserved word, reserved for future use.

CALLSTRIKECOUNT (Reserved Word)

Disclaimer

CALLSTRIKECOUNT is an OptionStation reserved word, reserved for future use.

CallVolume (Reserved Word)

Disclaimer

Returns the daily total Call option volume of the options for an underlying stock, index, or future. Allows historical bars back reference.

Example

```
   Value1 = CallVolume;
```

Cancel (Reserved Word)

Disclaimer

This reserved word is used to cancel an alert; it turns off any alerts triggered during the cur rent bar.

Remarks

- Cancel must always be followed by "Alert" in EasyLanguage code.
- Alerts are only generated for the last bar.

Example

If you wanted to cancel any alerts that have previously been enabled within the code based on certain criteria, you could use the following syntax:

```
If {Your Criteria Here} Then Cancel Alert;
```

Additional Example

If you write an indicator with two alert criteria, but you only want the alert to be triggered after 11:00am, you can use the following statements:

```
If Close Crosses Over Average(Close,10) Then
Alert("Average Cross Over");

If Volume > Average(Volume, 10) Then
Alert("Volume Spike");

If Time <= 1100 Then
 Cancel Alert;
```

If an alert is triggered by either one of the Alert statements, it is turned off by the Cancel Alert statement unless it is after 11:00am. Once it is after 11:00am, the alert is triggered when either Alert statement is true.

Category (Reserved Word)

Disclaimer

Returns a numeric expression identifying the type of symbol (such as stock, future, or index).

```
Value1 = Category ;
```

You must assign the reserved word to a numeric variable in order to obtain the number representing the type of symbol. The possible numbers are:

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
13	CPC Symbol
14	Composite

Ceiling (Reserved Word)

Disclaimer

Returns the lowest integer greater than the specified `Num`.

```
Ceiling(Num)
```

Where `Num` is a a numeric expression to be used in the calculation.

Examples

```
   Ceiling(4.5)
```

returns a value of 5.

```
   Ceiling(-1.72)
```

returns a value of -1.

CHAR (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

CheckAlert (Reserved Word)

Disclaimer

This reserved word determines whether or not the current bar is the last bar on the price chart (or grid) and whether or not the alert is enabled for the indicator or study.

When the alert is enabled and it is the last bar on the chart (or grid), this reserved word returns a value of True. This reserved word will return a value of False for all other bars on the price chart, and on the last bar of the price chart if the alert is not enabled.

Remarks

- CheckAlert differs from AlertEnabled by only checking the value in the Properties tab of the study to see if alerts are enabled on the last bar.
- Alerts are only generated for the last bar.

Example

CheckAlert will return True if alerts have been enabled and the bar is the last bar.

Additional Example

the following statements can be used to trigger an alert when the volume is twice the average volume, and to display the ratio between the current volume and the average. Because CheckAlert is used, the calculations are ignored for all historical bars as well as when the alert is not enabled.

```
If CheckAlert Then Begin
Value1 = Volume / Average(Volume, 10);
If Volume >= 2 * Average(Volume, 10) Then
 Alert ("Volume is" + NumtoStr(Value1, 2) );
End ;
```

CheckCommentary (Reserved Word)

Disclaimer

Returns True if the user clicks on a chart with the Analysis Commentary pointer on the specified bar. False is returned if the pointer has not been inserted, or if the pointer was inserted on a different bar.

Example

```
CheckCommentary
```

will return True if the Analysis Commentary Tool has been inserted for the specified bar.

Additional Example

If you only wanted code to be evaluated for the bar where the user had inserted the Analysis Commentary Tool, you could use the following syntax:

```
If CheckCommentary Then Begin
 {Your Code Here}
End;
```

CLEARDEBUG (Reserved Word)

Disclaimer

Clears the contents of the Output Debug window.

CLEARPRINTLOG (Reserved Word)

Disclaimer

CLEARPRINTLOG clears the Print Log in the EasyLanguage Output Bar.

Close (Reserved Word)

Disclaimer

Reserved word used to return the last price of the specified time increment or group of ticks.

Examples

```
Close of 1 bar ago
```

returns the Close price of the previous bar.

```
Average(Close, 10)
```

returns the Average of the last 10 Close prices.

Additional Example

To check that the last two bars have Close prices higher than the previous bar, the following language can be used in a ShowMe study:

```
If Close > Close[1] and Close[1] > Close[2] then
 Plot1(High, "ClosedUp");
```

COARSE (Reserved Word)

Disclaimer

Reserved for future use. COARSE specifies a coarse automatic optimization value for function inputs.

Commentary (Reserved Word)

Disclaimer

This reserved word sends the expression (or list of expressions) to the Analysis Commentary window for whatever bar is selected on the price chart.

You can use this reserved word multiple times, but it does not add a carriage return after the expression or list of expressions.

```
Commentary("My Expression") ;
```

`My Expression` is the numerical, text string or true/false expression that is to be sent to the Analysis 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") ;
```

You can use the `Commentary` reserved word multiple times, and it will not include a carriage return at the end of each expression (or list of expressions) sent. In order to generate a message that includes carriage returns, you will have to use either the NewLine or CommentaryCL reserved word.

Additional Examples

To include a carriage return in your Commentary, use the reserved word NewLine as a Commentary expression where needed. You can also use the reserved word CommentaryCL instead.

For example, the following statements produce the commentary in the example above:

```
   Commentary("This is one ");
   Commentary("line of commentary");
```

To include line breaks in the commentary, the reserved word NewLine needs to be used. For example, the following statements produce two lines of commentary text:

```
   Commentary("The 10-bar avg of the close", NewLine);
   Commentary(" is:", Average(Close, 10));
```

CommentaryCL (Reserved Word)

Disclaimer

This reserved word sends the expression (or list of expressions) to the Analysis Commentary window for whatever bar is selected by the Analysis Commentary pointer.

You can use this reserved word multiple times, and it will include a carriage return at the end of each expression (or list of expressions) sent.

```
CommentaryCL("My Expression") ;
```

`MyExpression` is a single or a comma-separated list of numeric, text string, or true/false expressions that are sent to the Analysis 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") ;
```

You can use the `CommentaryCL` reserved word multiple times, and it will include a carriage return at the end of each expression (or list of expressions) sent. In order to generate a message that does not include carriage returns, use the reserved word Commentary.

CommentaryEnabled (Reserved Word)

Disclaimer

This reserved word returns a value of True only when the Analysis 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 EasyLanguage to perform commentary-related calculations only when the Analysis Commentary window is open.

```
CommentaryEnabled
```

The difference between `CommentaryEnabled` and `AtCommentaryBar` is that `CommentaryEnabled` returns a value of True for ALL bars when the Analysis Commentary window is open, while the `AtCommentaryBar` returns a value of True only for the bar clicked with the Analysis Commentary pointer.

For example, the following statements calculate a cumulative advance/decline line to be displayed in the Analysis Commentary window:

Example

`CommentaryEnabled` will return True if the Analysis Commentary Tool has been applied to the chart.

Additional Example

```
If CommentaryEnabled Then Begin
 If Close > Close[1] Then
  Value1 = Value1 + Volume
 Else
  Value1 = Value1 - Volume;
 Commentary("The value of the A/D line is: ", Value1);
End;
```

Commission (Reserved Word)

Disclaimer

Returns the commission setting of the applied strategy.

Remarks

This reserved word can only be used in the evaluation of strategies.

Example

`Commission` returns a value of 17.50 if the commission has been set to 17.50.

CommodityNumber (Reserved Word)

Disclaimer

Retained for backward compatibility.

ComputerDateTime (Reserved Word)

Disclaimer

This reserved word returns a double-precision decimal DateTime value for the current computer time.

```
ComputerDateTime();
```

Examples

`Value1 = ComputerDateTime();` returns the value of your current computer's date/time in DateTime format.

Condition1-99 (Reserved Word)

Disclaimer

This reserved word refers to pre-declared variables in EasyLangauge that can be used to store a true-false condition from a calculation or an expression.

A variable is a name that represents a stored true-false value that was assigned from another condition or calculation so that you can reference it later in your code. In EasyLanguage, you can think of a condition variable as storage containers that hold a true-false value.

Note EasyLanguage provides you with 100 pre-declared numerical variables (Value0 through Value 99) and 100 pre-declared true/false variables (Condition0 through Condition99).

Const (Reserved Word)   same as Consts

Disclaimer

The reserved word `Const` (or `Consts`) is used to specify the name of a user-declared constant, its value, and optional data type. This must be done before a user-declared constant can be used in an assign statement or formula. Multiple constant names may be declared using a single `Const` statement where the names are separated by commas.

Syntax-Common

```
Const: Name(Value) ;
```

Where `Name` is the unique name of the user-declared constant, and `Value` is either a Numeric, True/False or String value that sets the value of the constant.

Syntax-Complete

```
Const: <DataType> VarName(ConstantValue) <, <DataType> VarName(ConstantValue)> ;
```

- `Const` is the constant declaration statement that may alternately be typed as `Consts`.
- `DataType` is the optionally supplied data type (float, double, int, bool, string) of the constant
- `VarName` is the user specified constant name that can be up to 20 characters long and must begin with an alphabetic character (names are not case sensitive)
- `ConstantValue` is the constant value.

Remarks

A constant name can contain up to 20 alphanumeric characters plus the period (.) and the underscore (_).

A constant name can not start with a number or a period (.).

The value of a constant is declared by a number in parenthesis after the input name.

Multiple constants names may be declared using a single `Const` statement where the names are separated by commas.

The use of the reserved words `Const` and `Consts` is identical.

Examples

```
   Const: Pi(3.1416);
```

Declares the constant `Pi` and sets the value as specified where the type is set based on the value.

```
   Consts: int MyValue(1025), float MyFactor(27.3618);
```

Declares the constant `MyValue` as an integer value of 1025 and `MyFactor` as a float value of 27.3618.

Continue (Reserved Word)

Disclaimer

The Continue reserved word transfers control to the end of a loop, bypassing any statements between it and the last statement of the loop.

Syntax

```
Continue;
```

Remarks

When a Continue statement is encountered in a Repeat loop, the Until expression is evaluated and the next iteration of the loop proceeds if needed.   In a For loop or While loop, reaching the End of the loop causes the loop to re-evaluate.

Example

In the following example, the Print statement before Continue is executed 3 times, but the Print statement after Continue is never evaluated. When the loops until condition is true and the loop exits, the Print statement following the loop is executed. The resulting output is what would appear in the Print Log of the EasyLanguage Output Bar.

```
Repeat
  i = i+1;
 Print("Before Continue");
Continue;
Print("This message never prints");
Until (i >= 3);
Print("After the loop");
```

Output (to the Print Log)

```
Before Continue
Before Continue
Before Continue
After the loop
```

Contract (Reserved Word)

Disclaimer

Reserved word used in conjunction with a numeric value specifying the number of units to trade within a trading strategy.

```
Contract
```

Remarks

`Contract` is normally used in a Buy, Sell, or Exit statement.

Note With Forex symbols, 1 contract refers to 1 lot (typically 100,000 units).

Examples

```
Buy 1 contract next bar at market
```

generates an order to b uy one contract at the open of the next bar.

```
SellShort 1 contract next bar at market
```

generates an order to sell one contract at the open of the next bar.

Additional Example

To exit from only 1 contract when you have multiple long positions, write:

```
Sell 1 contract next bar at market;
```

ContractMonth (Reserved Word)

Disclaimer

Returns the delivery (expiration) month of an option or position leg.

```
ContractMonth
```

Example

Plots an indicator line only when the contract month matches the month for a specified date.

```
If Month(Date) = ContractMonth then
      Plot1(Close);
```

ContractProfit (Reserved Word)

Disclaimer

Calculates the Position Profit per contract or share. `ContractProfit` divides the current position profit by the number of outstanding shares or contracts for the position profit for a single share or contract.

```
ContractProfit
```

Example

`Value1` will hold the position profit of 1 contract/share for the current position.

```
      Value1 = ContractProfit;
```

Additional Example

If you wanted to exit a long position after the profit per share reached $25, you could use the following syntax:

```
  If ContractProfit >= 25 Then
   Sell This Bar on Close;
```

Contracts (Reserved Word)

Disclaimer

Reserved word used in conjunction with a numeric value specifying the number of units to trade within a trading strategy.

Remarks

The reserved word `Contracts` is normally used in a Buy, Sell, or Exit statement.

Note With Forex symbols, 1 contract refers to 1 lot (typically 100,000 units).

Examples

```
Buy 5 contracts next bar at market
```

generates an order to buy five contracts at the open of the next bar.

```
SellShort 3 contracts next bar at market
```

generates an order to sell three contracts at the open of the next bar.

Additional Example

To exit from only 1 contract when you have multiple long positions, write:

```
Sell 1 contracts next bar at market;
```

ContractSize (Reserved Word)

Disclaimer

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

ContractYear (Reserved Word)

Disclaimer

Returns the delivery (expiration) year of an option or position leg.

```
ContractYear
```

Example

Plots an indicator line only when the contract year matches the year for a specified date.

```
       If Year(Date) = ContractYear then
              Plot1(Close);
```

Cosine (Reserved Word)

Disclaimer

Returns the cosine value of the specified number of degrees.

```
Cosine(Num);
```

Where `Num` is a numeric expression representing the number of degrees for which you want the cosine value.

Remarks

The cosine is a trigonometric function that for an acute angle is the ratio between the leg adjacent to the angle when it is considered part of a right triangle and the hypotenuse.

`Num` should be the number of degrees in the angle.

Examples

`Cosine(45)` returns a value of 0.7071.

`Cosine(72)` returns a value of 0.3090.

COST (Reserved Word)

Disclaimer

COST is used in OptionStation to return the value of the cost of establishing a leg or position.

Example

Cost of Position or Cost of Leg.

Cotangent (Reserved Word)

Disclaimer

Returns the cotangent value of the specified `Num` of degrees.

`CoTangent(Num)`

Where `Num` is a numeric expression to be used in the calculation.

Remarks

The cotangent is a trigonometric function that is equal to the cosine divided by the sine.

`Num` should be the number of degrees in the angle.

Examples

`CoTangent(45)` returns a value of 1.0.

`CoTangent(72)` returns a value of 0.3249.

CreateLeg (Reserved Word)

Disclaimer

Used only in OptionStation Search Strategies to create an option position leg for identification during the OptionStation Search process.

An option leg can be identified as a Call, Put, or Asset type, or Future type, and the number of base units that make up the position.

```
CreateLeg(Units, LegType);
```

`Units` is a numeric expression representing the number of base units that make up the leg position.

`LegType` can be identified using the reserved word `Call`, `Put`, `Asset type`, or `Future type`

Example

```
CreateLeg( -1, Call ) ;
CreateLeg( 100, AssetType ) ;
```

Cross (Reserved Word) same as Crosses

Disclaimer

The word Cross or Crosses is used to check for the crossing of two values.

```
Cross  or  Crosses
```

Remarks

Cross or Crosses is always followed by Above, Below, Over, or Under.

Examples

```
If Plot1 does Cross Above Plot2 Then...
```

Cross is used here to determine if the value of Plot1 crosses above the value of Plot2 on the bar under consideration.

```
If Value1 Crosses Above Value2 Or Value1 Crosses Below Value2 Then...
```

Crosses is used here to determine if Value1 crosses above or below Value2 on the bar under consideration.

CrossesAbove uses the following logic:

```
Inputs: lineA(numeric), lineB(numeric);
Vars: counter(0);

Condition1 = lineA > lineB;
Condition2 = lineA[1] = lineB[1];

Counter = 1;
While condition2 and counter < maxbarsback
 Begin
  Condition2 = lineA[counter] = lineB[counter];
  If condition2 then counter = counter + 1;
 End;
Condition3 = lineA[counter] < lineB[counter];
CrossesAbove = condition1 and condition3;
```

Current (Reserved Word)

Disclaimer

Reserved for future use.

Current_Ratio (Reserved Word)

Disclaimer

Retained for backward compatibility.

CurrentBar (Reserved Word)

Disclaimer

Returns the number of the bar currently being evaluated.

Each bar on a chart (after the number of bars specified by the **Maximum number of bars referenced by a study,** known as `MaxBarsBack`) is assigned a number, which is incremented by 1 with each successive bar. For example, if your `MaxBarsBack` is set to 10, the 11th bar is `CurrentBar` number 1, the 12th bar is `CurrentBar` number 2, and so on.

Remarks

`CurrentBar` can only be used to return the number of the current bar, for example, you cannot use:

```
CurrentBar[n]
```

… to obtain the bar number of the bar `n` bars ago. However, you can obtain the number of the bar `n` bars ago (for example, 5) by using:

```
CurrentBar - 5
```

Also, the `CurrentBar` reserved word is the same as the user function BarNumber. The only difference is that you can use the `BarNumber` function to reference past bars:

```
BarNumber[5]
```

Examples

You can use `CurrentBar` to determine how long ago a particular condition occurred:

```
   If Condition1 then
    Value1 = CurrentBar;
   If CurrentBar > Value1 then
    Value2 = CurrentBar – Value1;
```

`Value2` would hold the number of bars ago `Condition1` occurred.

CurrentContracts (Reserved Word)

Disclaimer

Returns the number of contracts in the current position (absolute value.)

Remarks

This function can only be used in the evaluation of strategies.

Example

`CurrentContracts` returns a value of 1 if the strategy is currently long 1 contract.

CurrentDate (Reserved Word)

Disclaimer

This reserved word returns a numeric value representing the EasyLanguage date (YYYM MDD format) corresponding to the date and time of your computer.

Examples

`CurrentDate` returns a value of 991016 on October 16, 1999.

`CurrentDate` returns a value of 1011220 on December 20, 2001.

Additional Example

To have a trading strategy, analysis technique, or function perform its calculations only be fore January 1, 2000 (or any other date for that matter), you can write:

```
If CurrentDate < ELDate(2000, 01, 01) Then Begin
 { EasyLanguage instruction(s) }
End;
```

CurrentEntries (Reserved Word)

Disclaimer

Returns the number of entries currently open within a position.

Remarks

This function can only be used in the evaluation of strategies.

Example

`CurrentEntries` returns a value of 3 if the strategy has made 3 entries in the current open position.

CurrentOpenInt (Reserved Word)

Disclaimer

Returns a numeric expression representing the last known open interest for the symbol.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

CurrentShares (Reserved Word)

Disclaimer

Returns the number of shares in the current position (absolute value.)

Remarks

This function can only be used in the evaluation of strategies.

Example

`CurrentShares` returns a value of 100 if the strategy is currently long 100 shares.

CurrentTime (Reserved Word)

Disclaimer

This reserved word returns a numeric value representing the EasyLanguage time (HHMM format) corresponding to the time of your computer.

Examples

`CurrentTime` returns a value of 1718 at 5:18 pm.

`CurrentTime` returns a value of 0930 at 9:30 am.

Additional Example

To have a trading strategy, analysis technique, or function perform its calculations only if it is before 2:00pm, you can write:

```
If CurrentTime < 1400 Then Begin
{ EasyLanguage instruction(s) }
End;
```

CustomerID (Reserved Word)

Disclaimer

Returns the User ID number of the person to whom the number is registered.

Remarks

Each customer has a unique customer identification number that can be used to identify his/her program

Examples

`CustomerID` returns the customer identification number.

Cyan (Reserved Word)

Disclaimer

Sets the plot color or background color to Cyan.

```
Cyan
```

Remarks

There are 16 colors available in EasyLanguage. For more information, see EasyLanguage Colors and Their Corresponding Numeric Values.

Examples

```
Plot1(Value1, "Test", Cyan)
```

plots `Value1` with the name `Test`, and sets the color of `Plot1` to `Cyan`.

```
TL_SetColor(1, Cyan)
```

sets the color of a TrendLine with a reference number of 1 to `Cyan`.

D (Reserved Word)

Disclaimer

Shortcut notation that returns the closing Date of the bar referenced.

Remarks

Anytime the Date of a bar is needed, the letter `D` can be used in an equivalent fashion.

`D` returns the date in YYMMDD format.

Examples

`D` returns 1000107 if the Date of the bar is January 7th, 2000.

`D` returns 990412 if the Date of the bar is April 12th, 1999.

Additional Example

To check the Date of the previous bar write the following code:

```
D of 1 bar ago
```

DailyClose (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

DailyHigh (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

DailyLimit (Reserved Word)

Disclaimer

Retained for backward compatibility.

DailyLow (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

DailyOpen (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

DailyOpen (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

DailyTradesDN (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

DailyTradesUC (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

DailyTradesUP (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

DailyVolume (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the total trade volume of the trading day.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

DailyVolumeDN (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the down volume for the current trading day.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

DailyVolumeUC (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the unchanged volume for the current trading day.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

DailyVolumeUp (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the up volume for the current trading day.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

DarkBlue (Reserved Word)

Disclaimer

Sets the plot color or background color to Dark Blue.

```
DarkBlue
```

Remarks

There are 16 colors available in EasyLanguage. For more information, see EasyLanguage Colors and Their Corresponding Numeric Values.

Examples

```
   Plot1(Value1, "Test", DarkBlue)
```

plots `Value1` with the name `Test`, and sets the color of `Plot1` to `DarkBlue`.

```
   TL_SetColor(1, DarkBlue)
```

sets the color of a TrendLine with a reference number of 1 to `DarkBlue`.

DarkBrown (Reserved Word)

Disclaimer

Sets the plot color or background color to Dark Brown.

```
DarkBrown
```

Remarks

There are 16 colors available in EasyLanguage. For more information, see EasyLanguage Colors and Their Corresponding Numeric Values.

Examples

```
  Plot1(Value1, "Test", DarkBrown)
```

plots `Value1` with the name `Test`, and sets the color of `Plot1` to `DarkBrown`.

```
  TL_SetColor(1, DarkBrown)
```

sets the color of a TrendLine with a reference number of 1 to `DarkBrown`.

DarkCyan (Reserved Word)

Disclaimer

Sets the plot color or background color to Dark Cyan.

```
DarkCyan
```

Remarks

There are 16 colors available in EasyLanguage. For more information, see EasyLanguage Colors and Their Corresponding Numeric Values.

Examples

```
   Plot1(Value1, "Test", DarkCyan)
```

plots `Value1` with the name `Test`, and sets the color of `Plot1` to `DarkCyan`.

```
   TL_SetColor(1, DarkCyan)
```

sets the color of a TrendLine with a reference number of 1 to `DarkCyan`.

DarkGray (Reserved Word)

Disclaimer

Sets the plot color or background color to Dark Gray.

```
DarkGray
```

Remarks

There are 16 colors available in EasyLanguage. For more information, see EasyLanguage Colors and Their Corresponding Numeric Values.

Examples

```
Plot1(Value1, "Test", DarkGray)
```

plots Value1 with the name Test, and sets the color of Plot1 to DarkGray.

```
TL_SetColor(1, DarkGray)
```

sets the color of a TrendLine with a reference number of 1 to DarkGray.

DarkGreen (Reserved Word)

Disclaimer

Sets the plot color or background color to Dark Green.

```
DarkGreen
```

Remarks

There are 16 colors available in EasyLanguage. For more information, see EasyLanguage Colors and Their Corresponding Numeric Values.

Examples

```
Plot1(Value1, "Test", DarkGreen)
```

plots `Value1` with the name `Test`, and sets the color of `Plot1` to `DarkGreen`.

```
TL_SetColor(1, DarkGreen)
```

sets the color of a TrendLine with a reference number of 1 to `DarkGreen`.

DarkMagenta (Reserved Word)

Disclaimer

Sets the plot color or background color to Dark Magenta.

```
DarkMagenta
```

Remarks

There are 16 colors available in EasyLanguage. For more information, see EasyLanguage Colors and Their Corresponding Numeric Values.

Examples

```
Plot1(Value1, "Test", DarkMagenta)
```

plots `Value1` with the name `Test`, and sets the color of `Plot1` to `DarkMagenta`.

```
TL_SetColor(1, DarkMagenta)
```

sets the color of a TrendLine with a reference number of 1 to `DarkMagenta`.

DarkRed (Reserved Word)

Disclaimer

Sets the plot color or background color to Dark Red.

```
DarkRed
```

Remarks

There are 16 colors available in EasyLanguage. For more information, see EasyLanguage Colors and Their Corresponding Numeric Values.

Examples

```
Plot1(Value1, "Test", DarkRed)
```

plots `Value1` with the name `Test`, and sets the color of `Plot1` to `DarkRed`.

```
TL_SetColor(1, DarkRed)
```

sets the color of a TrendLine with a reference number of 1 to `DarkRed`.

Data (Reserved Word)

Disclaimer

Reserved word used to reference information from a specified data stream.

Remarks

`Data` is normally used with a number between 1-50 that allows the specification or which data set is being referred to in terms of price values and functions calculations.

Examples

`Close of Data3` returns the Close price of Data stream 3.

`Low of Data10` returns the Low price of Data stream 10.

DataCompression (Reserved Word)

Disclaimer

Returns a number indicating the interval setting of the price data an analysis technique is applied to.

Remarks

The number returned is based on the data interval of the price data. `DataCompression` will return:

0 = TickBar
1 = Intraday
2 = Daily
3 = Weekly
4 = Monthly
5 = Point & Figure

Examples

`DataCompression` returns 0 when applied to price data based on tick interval

`DataCompression` returns 2 when applied to price data based on daily interval

Additional Example

To assure that a statement is executed only on a daily chart we can write:

```
If DataCompression = 2 then Sell next bar at market;
```

DataInUnion (Reserved Word)

Disclaimer

Reserved for future use.

Date (Reserved Word)

Disclaimer

This reserved word returns a numeric expression representing the EasyLanguage date of the closing price of the bar being analyzed. The date is an EasyLanguage date, so it is a numeric expression of the form YYYMMDD, where YYY is years since 1900, MM is the month, and DD is the day of the month.

Remarks

`Date` returns a numeric value in YYYMMDD format.

Examples

`Date` returns 1000107 if the day is January 7th, 2000.

`Date` returns 990412 if the day is April, 12th, 1999.

Additional Example

`Date` can be used to restrict strategies to certain trading days.

```
If Date < 990101 then buy this bar on close;
```

Limits a buy order to take place only on dates before 1999.

DateTimeToString (Reserved_Word)

Disclaimer

This reserved word returns a string expression representing the specified DateTime value.

```
DateTimeToString(dDateTime);
```

Where `dDateTime` is a double-precision decimal expression representing the Julian Date (since 1900) and time (since midnight).

Remarks

If a specific date is entered for the numeric expression, it must be a valid Julian date between January 1, 1900 and February 28, 2150, expressed in DateTime format.

Examples

`strVar = DateTimeToString(37561.61200);` returns a value of "11/1/2002 2:41:17 PM" for the specified DateTime of 37561.61200.

DateToJulian (Reserved Word)

Disclaimer

This reserved word returns a numeric expression representing the Julian Date equivalent to the specified EasyLanguage date.

```
DateToJulian(cDate);
```

Where `cDate` is a numeric expression representing the EasyLanguage calendar date (YYYMMDD format, 1999 = 99, 2001 = 101).

Remarks

If a specific date is entered for the numeric expression, it must be a valid date between January 1, 1900 and February 28, 2150, expressed in YYMMDD or YYYMMDD format.

Examples

`DateToJulian(980804)` returns a value of 36011 for the date August 4, 1998.

`DateToJulian(991024)` returns a value of 36457 for the date October 24, 1999.

DateToString (Reserved_Word)

Disclaimer

This reserved word returns a string expression representing only the date portion of the specified DateTime value.

```
DateToString(dDateTime);
```

Where `dDateTime` is a double-precision decimal expression representing the Julian Date (since 1900) and time (since midnight).

Remarks

If a specific date is entered for the numeric expression, it must be a valid Julian date between January 1, 1900 and February 28, 2150, expressed in DateTime format.

Examples

`strVar = DateToString(37561.61200);` returns a value of "11/1/2002" for the specified DateTime of 37561.61200.

Day (Reserved Word)

Disclaimer

Reserved word used for backwards compatibility.

Remarks

`Day` has been replaced with the reserved word, Bar.

Examples

```
Close of 1 day Ago
```

returns the `close` price of the previous bar.

```
BuyToCover this day on close
```

orders a trading strategy to exit all short positions on the `close` of the current bar.

Additional Example

```
Buy ("Order Name") next day at open
```

will buy during the formation of the next bar if a price meets the open price of the bar.

DayFromDateTime (Reserved Word)

Disclaimer

Returns the numeric day of the month for the specified DateTime value.

```
DayFromDateTime(dDateTime);
```

Where `dDateTime` is a double-precision decimal expression representing the Julian Date (since 1900) and time (since midnight).

Remarks

If a specific date is entered for the numeric expression, it must be a valid Julian date between January 1, 1900 and February 28, 2150, expressed in DateTime format.

Examples

`Value1 = DateFromDateTime(37564.61200);` returns a day value of 4 for the specified DateTime of 37564.61200 (11/4/2002 2:41:17 PM).

DayOfMonth (Reserved Word)

Disclaimer

Returns the day of month for the specified calendar date.

```
DayOfMonth(cDate);
```

Where `cDate` is a numeric expression representing the six or seven digit calendar date in the format YYMMDD or YYYMMDD respectively (1999 = 99, 2001 = 101).

Examples

```
   DayOfMonth(980804)
```

returns a value of 4 for the date August 4, 1998.

```
   DayOfMonth(1011024)
```

returns a value of 24 for the date October 24, 2001.

DayOfWeek (Reserved Word)

Disclaimer

Returns the day of week for the specified calendar date. (0 = Sun, 6 = Sat).

```
DayOfWeek(cDate);
```

Where `cDate` is a numeric expression representing the six or seven digit EasyLanguage calendar date in the format YYMMDD or YYYMMDD respectively. (1999 = 99, 2001 = 101)

Examples

```
  DayOfWeek(980804)
```

returns a value of 2 because August 4, 1998 is a Tuesday.

```
  DayOfWeek(1011024)
```

returns a value of 3 because October 24, 2001 is a Wednesday.

DayOfWeekFromDateTime (Reserved Word)

Disclaimer

Returns the day of week (0 = Sun, 6 = Sat) from a specified DateTime value.

```
DayOfWeekFromDateTime(dDateTime);
```

Where `dDateTime` is a double-precision decimal expression representing the Julian Date (since 1900) and time (since midnight).

Remarks

If a specific date is entered for the numeric expression, it must be a valid Julian date between January 1, 1900 and February 28, 2150, expressed in DateTime format.

Examples

`Value1 = DayOfWeekFromDateTime(37564.61200);` returns a value of 1 (Monday) for the specified DateTime of 37564.61200 (11/4/2002 2:41:17 PM).

Days (Reserved Word)

Disclaimer

Reserved word used for backwards compatibility.

Remarks

`Days` has been replaced with the reserved word, Bar.

Example

```
Close of 5 Days Ago
```

returns the `Close` price of the bar 5 bars previous to the current bar.

Default (Reserved Word)

Disclaimer

This word is used in plot statements to set one of its styles (for example, color) to the default value.

Example:

```
Plot1(Value1, "Plot1", Default, Default, 5);
```

`Default` is used here to designate the items as the user set default values.

DefineCustField (Reserved Word)

Disclaimer

This word has been reserved for future use.

DEFINEDLLFUNC (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

DeliveryMonth (Reserved Word)

Disclaimer

Reserved word used for contracts that expire which returns the month of expiration.

Remarks

The value returned represents any of the 12 calendar months according to the following table:

1 January
2 February
3 March
4 April
5 May
6 June
7 July
8 August
9 September
10 October
11 November
12 December

Examples

`DeliveryMonth` will return 6 if the technique is applied to the S&P June contract.

`DeliveryMonth` will return 12 if the technique is applied to the Treasury Bond December contract.

Additional Example

You can exit from a trade on the first day of the month prior to expiration using:

```
If DeliveryMonth = Month(Date) Then
 Sell This Bar on Close;
```

DeliveryYear (Reserved Word)

Disclaimer

Reserved word used for contracts that expire which returns the year of expiration.

Remarks

The value returned by `DeliveryYear` is a four-digit year.

Examples

`DeliveryYear` will return 1998 if the technique is applied to the S&P June ‘98 contract.

`DeliveryYear` will return 1999 if the technique is applied to the Treasury Bond March ’99 contract.

Additional Example

You can exit from a trade on the first day of the year prior to expiration using:

```
If DeliveryYear = Year(Date) Then
 Sell This Bar on Close;
```

DELTA (Reserved Word)

Disclaimer

DELTA is used in OptionStation to return the risk value Delta of an option or position.

Description (Reserved Word)

Disclaimer

Returns a string containing the description of the symbol if it is available. If the symbol has no description, this reserved word will return a blank string ("").

```
    TextString = Description;
```

You must assign the reserved word to a string variable in order to obtain the description. EasyLanguage does not provide pre-declared string variables; you must declare the string variable before you can assign a reserved word to it.

Dividend (Reserved Word)

Disclaimer

Retained for backward compatibility.

Dividend_Yield (Reserved Word)

Disclaimer

Retained for backward compatibility.

DividendCount (Reserved Word)

Disclaimer

Retained for backward compatibility.

DividendDate (Reserved Word)

Disclaimer

Retained for backward compatibility.

DividendFreq (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

DividendTime (Reserved Word)

Disclaimer

Retained for backward compatibility.

DivYield (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the yield of a bond.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

Does (Reserved Word)

Disclaimer

DOES is a Skip Word that performs no function within your code. It is used within a statement only to make the EasyLanguage code more readable.

Remarks

Skip words in EasyLanguage are not necessary, however, they sometimes make it easier to understand the purpose of the code.

Syntax Coloring in the EasyLanguage PowerEditor is dark green for all skip words by default.

Example

```
If the Close does cross over 100 Then
      Alert;
```

The word "`does`" is not necessary and the code functions the same way regardless of its presence.

DOUBLE (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

DoubleQuote (Reserved Word)

Disclaimer

DoubleQuote will embed a double-quote (") character in a string.

DownTicks (Reserved Word)

Disclaimer

Returns the number of ticks on a bar whose value is lower than the tick immediately preceding it.

```
DownTicks
```

Remarks

Returns the down volume of a bar when Trade Volume is used for the chart. See EasyLanguage Reserved Words Related to Ticks, Volume & Open Interest for more information.

Examples

DownTicks returns 5 if there were 5 ticks on a bar whose value was lower than the tick immediately preceding it.

DownTicks returns 12 if there were 12 ticks on a bar whose value was lower than the tick immediately preceding it.

Additional Example

To check if a bar of data appears to reflect a steady downturn, compare DownTicks to UpTicks:

```
   Value1 = DownTicks - UpTicks;
```

DownTo (Reserved Word)

Disclaimer

This word is used as a part of a `For` statement where the execution values will be decreasing to a finishing value.

`DownTo`

Remarks

`DownTo` will always be placed between two arithmetic expressions.

Example

```
For Value5 = Length DownTo 0 Begin
 {Your Code Here}
End;
```

`DownTo` is used here to indicate that for each value of `Value5` from Length down to zero, the following `Begin`… `End` loop will be executed.

DWORD (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

EasyLanguageVersion (Reserved Word)

Disclaimer

Returns the EasyLanguage version currently installed.

Remarks

The EasyLanguage version for TradeStation is 5.1.

Examples

If you wanted to only display an indicator when the user had EasyLanguage 4.0 or later installed on the machine, you could use the following syntax:

```
If EasyLanguageVersion >= 4.0 Then
 Plot1(Value1, "Indicator");
```

EL_DateStr (Reserved Word)

Disclaimer

EasyLanguage Date to String returns an 8-character date string in YYYYMMDD format.

`EL_DateStr(DD, MM, YYYY)`

Where `DD` is a numeric expression representing the day of the month.

`MM` is a numeric expression representing a month (for example, January = 01).

`YYYY` is a numeric expression representing a four-digit year.

Remarks

Returned string date is in YYYYMMDD format.

A four-digit year is required for the year Input.

Examples

`EL_DateStr(09, 05, 1999)` returns the string 19990509

`EL_DateStr(25, 11, 2004)` returns the string 20041125

ELDateToDateTime (Reserved Word)

Disclaimer

This reserved word returns a double-precision decimal expression representing the specified EasyLanguage date value.

```
ELDateToDateTime(eDate);
```

Where `eDate` is a numeric expression representing the date in EasyLanguage YYYMMDD format.

Remarks

If a specific date is entered, it must be a valid date between January 1, 1900 and February 28, 2150, expressed in EasyLanguage format.

Examples

`Value1 = ELDateToDateTime(1030425);` returns a value of 37736.00 for the specified date of April, 25, 2003..

Else (Reserved Word)

Disclaimer

This word is used to execute a series of code based on a condition that has returned a value of `False`.

`Else`

Remarks

`Else` can only be used following an `If… Then` statement.

Examples

```
If Condition1 Then
 {Your Code Line1}
Else Begin
 {Your Code Line2}
End;
```

`Else` is used here to begin the code that will be executed if `Condition1` returns a value of `False`. No semicolon should be used preceding the `Else` statement.

```
If Condition1 And Condition2 Then Begin
 {Your Code Line1};
 {Your Code Line2, etc.};
End
Else Begin
 {Your Code Line3};
 {Your Code Line4, etc.};
End;
```

`Else` is used here to begin the code that will be executed if either `Condition1` or `Condition2` return a value of `False`. Again, no semicolon should be used preceding the `Else` statement.

ELTimeToDateTime (Reserved Word)

Disclaimer

This reserved word returns a double-precision decimal expression representing the specified EL time value.

```
ELTimeToDateTime(eTime);
```

Where `eTime` is a numeric expression representing the time in EasyLanguage HHMM 24-hour format.

Remarks

If a specific time is entered, it must be a valid time in 24-hour military format.

Examples

`Value1 = ELTimeToDateTime(2152);` returns a value of .911111 that represents the specified EL time of 9:52pm.

EncodeDate (Reserved Word)

Disclaimer

This reserved word returns a DateTime value expression representing the three specified numeric date parameters (YYY, MM, DD).

```
EncodeDate(99,04,07);
```

Remarks

The specified numeric values must represent a valid date between January 1, 1900 and February 28, 2150 consisting of YYY, MM, DD.

Examples

`Value1 = EncodeDate(02,01,03)` returns a value of 37259.00 for the date January 3, 2002.

EncodeTime (Reserved Word)

Disclaimer

This reserved word returns a DateTime value expression representing the four specified numeric time parameters (Hours, Minutes, Seconds, Milliseconds).

```
EncodeTime(13,24,10,06);
```

Remarks

The specified numeric values must represent a valid time in 24-hour military format.

Examples

`Value1 = EncodeTime(13,24,10,06)` returns a value of 0.56 for 1:24:10.06 PM.

End (Reserved Word)

Disclaimer

This word is used to end a series of code that should be executed on the basis of an If… Then, If… Then… Else, For, or While statement.

End

Remarks

Each Begin must have a corresponding End.

A Begin… End statement is only necessary if using more than one line of code after an If… Then, If… Then… Else, For, or While statement.

Examples

```
If Condition1 Then Begin
 {Your Code Line1}
 {Your Code Line2}
End;
```

End is used here to exclude the execution of code beyond Line1 and Line2 as a part of the If… Then statement.

```
If Condition1 And Condition2 Then Begin
 {Your Code Line1};
 {Your Code Line2, etc.};
End
Else Begin
 {Your Code Line3};
 {Your Code Line4, etc.};
End;
```

End is used here twice to limit the execution to the two lines of each section that are intended based on Condition1. No semicolon should be used after the End preceding the Else statement. If… Then… Else is part of a continuous section of code and only the End that finishes the If… Then… Else statement should be followed by a semicolon.

Entry (Reserved Word)

Disclaimer

An optional Exit parameter used to reference a specific, named entry.

Remarks

Entry names are required.

Entry is exclusively used in an Exit condition to specify the Entry to exit from

Examples

```
Sell from entry ("MyTrade") next bar market;
```

Generates an order to exit the long position "MyTrade" on the first price of the next bar.

```
BuyToCover from entry ("MyTrade") this bar on close;
```

Generates an order to exit the short position "MyTrade" at the close of the current bar.

Additional Example

```
Sell from entry ("MyTrade") next bar at 75 Stop;
```

Generates an order to exit the long position "MyTrade" on the next bar at a price of 75 or lower.

EntryDate (Reserved Word)

Disclaimer

Returns the entry date for the specified position in the format YYYYMMDD.

```
EntryDate(Num)
```

Where `Num` is a numeric expression representing the number of positions ago (up to a maximum of ten).

Remarks

This function can only be used in the evaluation of strategies. It does not require an input, however, by using the input `Num`, you can obtain the specified value from a previous position, up to ten positions ago.

Example

`EntryDate(2)` might return a value of 981005 if the entry date of 2 positions ago was October 5, 1998.

EntryPrice (Reserved Word)

Disclaimer

Returns the entry price for the specified position.

```
EntryPrice(Num)
```

Where `Num` is a numeric expression representing the number of positions ago (up to a maximum of ten).

Remarks

This function can only be used in the evaluation of strategies. It does not require an input, however, by using the input `Num`, you can obtain the specified value from a previous position, up to ten positions ago.

Example

`EntryPrice(2)` might return a value of 101.19 as the entry price of 2 positions ago on a chart of Microsoft stock.

EntryTime (Reserved Word)

Disclaimer

Returns the entry time for the specified position in military time format HHMM.

`EntryTime(Num)`

Where `Num` is a numeric expression representing the number of positions ago (up to a maximum of ten).

Remarks

This function can only be used in the evaluation of strategies. It does not require an input, however, by using the input `Num`, you can obtain the specified value from a previous position, up to ten positions ago.

Example

`EntryTime(2)` might return a value of 1600 as the entry time of 2 positions ago on a daily chart of Microsoft stock.

EPS (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

EPS_PChng_Y_Ago (Reserved Word)

Disclaimer

Retained for backward compatibility.

EPS_PChng_YTD (Reserved Word)

Disclaimer

Retained for backward compatibility.

EPSCount (Reserved Word)

Disclaimer

Retained for backward compatibility.

EPSDate (Reserved Word)

Disclaimer

Retained for backward compatibility.

EPSEstimate (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

EPSTime (Reserved Word)

Disclaimer

Retained for backward compatibility.

ExchListed (Reserved Word)

Disclaimer

A quote field that returns a string expression representing the exchange under which the symbol is listed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

ExecOffset (Reserved Word)

Disclaimer

ExecOffset returns the local function execution bar-offset amount.

ExitDate (Reserved Word)

Disclaimer

Returns the exit date for the specified position in the format YYYMMDD.

```
        ExitDate(Num) ;
```

Where `Num` is a numeric expression representing the number of positions ago (up to a maximum of ten).

Remarks

This function can only be used in the evaluation of strategies. It does not require an input, however, by using the input `Num`, you can obtain the specified value from a previous position, up to ten positions ago.

Example

`ExitDate(2)` might return a value of 981022 if the exit date of 2 positions ago was October 22, 1998.

ExitPrice (Reserved Word)

Disclaimer

Returns the exit price for the specified position.

```
        ExitPrice(Num) ;
```

Where `Num` is a numeric expression representing the number of positions ago (up to a maximum of ten).

Remarks

This function can only be used in the evaluation of strategies. It does not require an input, however, by using the input `Num`, you can obtain the specified value from a previous position, up to ten positions ago.

Example

`ExitPrice(1)` might return a value of 107.750 as the exit price of the last position on a chart of Microsoft stock.

ExitTime (Reserved Word)

Disclaimer

Returns the exit time for the specified position in military time format HHMM.

```
        ExitTime(Num) ;
```

Where `Num` is a numeric expression representing the number of positions ago.

Remarks

This function can only be used in the evaluation of strategies. It does not require an input, however, by using the input `Num`, you can obtain the specified value from a previous position, up to ten positions ago.

Example

`ExitTime(3)` might return a value of 1050 as the exit time of 3 positions ago on an intraday chart of Microsoft stock.

ExpDate (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the EasyLanguage date of the expiration date for the symbol. The expiration date is defined by the expiration rule chosen for the symbol.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

ExpDate (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the Julian date (accurate to the second) of the expiration date for the symbol. The expiration date is defined by the expiration rule chosen for the symbol.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

ExpirationDate (Reserved Word)

Disclaimer

Returns the Date of Expiration of the asset to which a technique is applied.

Remarks

`ExpirationDate` returns a four-digit year.

EXPIRATIONSTYLE (Reserved Word)

Disclaimer

EXPIRATIONSTYLE is used in OptionStation to return a value specifying an option's expiration rule (0=American, 1=European).

ExpStyle (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the expiration style of an option. Where 0 is American and 1 is European. Debit positions return positive numbers and credit positions return negative numbers.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

ExpValue (Reserved Word)

Disclaimer

Returns the exponential value of the specified (Num).

```
ExpValue(Num)
```

Where `Num` is a numeric expression to be used in the calculation.

Examples

`ExpValue(4.5)` returns a value of 90.0171.

`ExpValue(6)` returns a value of 403.4288.

EXTERNAL (Reserved Word)

Disclaimer

EXTERNAL defines a DLL entry point to be called from EasyLanguage.

False (Reserved Word)

Disclaimer

This word is used as the value for an input or untrue condition.

```
False
```

Remarks

`False` can only be used in EasyLanguage as the value for a True/False input.

`False` is the value returned by an untrue or invalid condition such as, 1 < 0.

Examples

```
Input: Test(False);
```

would set the value of an input `Test` to a default value of `False`.

File (Reserved Word)

Disclaimer

Sends information to a specified file from a Print statement.

```
File("c:\windows\filename.txt")
```

Where "`c:\windows\filename.txt`" is a string expression that must be a valid path and file name encompassed in quotes.

Remarks

The name of the file must be string literal as variables are not permitted as file names.

Example

If you wanted to send the date, time, and close information from a chart to a new ascii file named mydata and located in the data directory by applying an indicator, you could use the following syntax:

```
   Print(File("c:\data\mydata.txt"), Date, Time, Close);
```

FileAppend (Reserved Word)

Disclaimer

Sends information to an existing ASCII file specified by the user and adds the information to the bottom of the file.

```
        FileAppend("str_Filename", "str_Text") ;
```

Where `str_Filename` is a string expression that is the path and file name for the existing ASCII text file to which you want to append the string of text. The path and file name should be enclosed in quotes.

`str_Text` is the string expression to be appended to the file.

Remarks

If the file does not exist, it will be created.

To send numeric expressions to a file, it must first be converted to a string expression using the `NumToStr` function.

Example

To create indicator that would send the date to an ASCII file every time the symbol gapped up 10% on the open, you would write:

```
If Open > High[1] * 1.1 Then
 FileAppend("c:\mydata.txt", "This symbol gapped up on " + NumToStr(Date, 0) +
NewLine);
```

FileDelete (Reserved Word)

Disclaimer

Deletes the specified file.

```
FileDelete("str_Filename");
```

str_Filename is a string expression that must be a valid path and file name encompassed in quotes.

Example

If you wanted to delete a file when an analysis technique was initially applied, you could use the following syntax:

```
If BarNumber = 1 Then
 FileDelete("c:\mydata.txt");
```

FINE (Reserved Word)

Disclaimer

Reserved for future use. FINE specifies a fine automatic optimization value for function inputs.

FirstNoticeDate (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by FND.

FirstNoticeDate (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by FNDEX.

FIRSTOPTION (Reserved Word)

Disclaimer

FIRSTOPTION returns true on the first option processed during on OptionStation core calculation event.

FLOAT (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

Floor (Reserved Word)

Disclaimer

Returns the highest integer less than the specified (Num).

`Floor(Num)`

Where `Num` is a numeric expression to be used in the calculation.

Examples

`Floor(4.5)` returns a value of 4.

`Floor(-1.72)` returns a value of -2.

FND (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the EasyLanguage date of the first notice of a futures contract. The first notice date is defined by the expiration rules.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

Example

```
If Date = FND then Alert("First Notice Expiration");
```

FND (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the Julian date (accurate to the second) of the first notice of a futures contract. The first notice date is defined by the expiration rules.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

Example

```
If Date = FND then Alert("First Notice Expiration");
```

For (Reserved Word)

Disclaimer

For loops are used to conduct a repeated line or lines of code for a specified number of intervals.

For

Remarks

For will always be followed by a variable name, the range to be repeated, and finally a Begin… End section containing the code to be executed based on the For expression.

Examples

```
For Value5 = Length To Length + 10 Begin
 {Your Code Here}
End;
```

For is used here to initiate a loop that will be executed for each value of Value5 from Length to Length plus 10.

```
Variables: Sum(0), Counter(0);
Sum = 0;
For Counter = 0 To Length - 1 Begin
 Sum = Sum + Price[Counter];
End;
```

For is used here to initiate the accumulation of the variable Sum for each value of Counter from 0 to Length minus one.

FormatDate (Reserved Word)

Disclaimer

This reserved word returns a formatted string expression representing the date portion of the specified DateTime value.

```
FormatDate("ddd MMM dd yy", dDateTime ));
```

Where "ddd" is the day of the week, "MMM is the month", "dd" is the day of the month, and "yy" is the year in the string format expression, and where. `dDateTime` is a double-precision decimal expression representing the Julian Date (since 1900) and time (since midnight)

Remarks

If a specific date is entered for the numeric expression, it must be a valid Julian date between January 1, 1900 and February 28, 2150, expressed in DateTime format.

Examples

`strVar =(FormatDate("ddd MMM dd yy",37561.61200));` returns a value of "Fri Nov 01 02" for the specified DateTime of 37561.61200.

`Print( FormatDate( "ddd MMM dd yy", ElDateToDateTime( Date ) ) );` prints the current date in the format "Day Month Date Year".

FormatTime (Reserved Word)

Disclaimer

This reserved word returns a formatted string expression representing the time portion of the specified DateTime value.

```
FormatTime("hh:mm:ss tt", dDateTime ));
```

Where "hh" is the hour, "mm" is the minutes", "ss" is the seconds, and "tt" is AM or PM in the string format expression, and where. `dDateTime` is a double-precision decimal expression representing the time (since midnight)

Remarks

If a specific time is entered for the numeric expression, it must be a valid time, expressed in DateTime format.

Examples

`strVar =(FormatTime("hh:mm:ss tt",37561.61200));` returns a value of "02:41:17 PM" for the specified DateTime of 37561.61200.

FracPortion (Reserved Word)

Disclaimer

Returns the fractional portion of the specified (Num).

`FracPortion(Num)`

Where `Num` is a numeric expression to be used in the calculation.

Examples

`FracPortion(4.5)` returns a value of 0.5.

`FracPortion(-1.72)` returns a value of –0.72.

FreeCshFlwPerShare (Reserved Word)

Disclaimer

Retained for backward compatibility.

Friday (Reserved Word)

Disclaimer

Returns the number 5 as the value for Friday.

```
Friday
```

Example

```
   Friday
```

…returns a value of 5 as the value for Friday.

From (Reserved Word)

Disclaimer

Reserved word used in an Exit statement to specify the name of a Long or Short entry.

```
From
```

Remarks

`From` is exclusively used in an Exit condition along with "`entry`", to specify the Entry to exit from

Examples

```
Sell from entry ("MyTrade") next bar market;
```

Generates an order to exit the long position "`MyTrade`" on the first price of the next bar.

```
BuyToCover from entry ("MyTrade") this bar on close;
```

Generates an order to exit the short position "`MyTrade`" at the close of the current bar.

Additional Example

```
Sell from entry ("MyTrade") next bar at 75 Stop;
```

Generates an order to exit the long position "`MyTrade`" on the next bar at a price of 75 or lower.

FUTURE (Reserved Word)

Disclaimer

FUTURE is an OptionStation reserved word, it is a modifier to other Future related reserve words.

Example

Close of Future;

FUTURETYPE (Reserved Word)

Disclaimer

FUTURETYPE is an OptionStation reserved word that returns TRUE if a position leg is determined to be a future.

G_Rate_EPS_NY (Reserved Word)

Disclaimer

Retained for backward compatibility.

G_Rate_Nt_In_NY (Reserved Word)

Disclaimer

Retained for backward compatibility.

G_Rate_P_Net_Inc (Reserved Word)

Disclaimer

Retained for backward compatibility.

GAMMA (Reserved Word)

Disclaimer

GAMMA is used in OptionStation to return the risk value Gamma of an option or position.

GetAccount (Reserved Word)

Disclaimer

Identifies the account for a location number in the list of accounts. This function can be used with GetNumAccounts to enumerate available equity and futures accounts.

Usage:

```
GetAccount(AccounLoc)
```

Return:

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

Inputs:

AccountLoc is an integer number with a value between 1 and the number returned by GetNumAccounts.

Example:

```
   vars: str("");
   str = GetAccount(1);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetAccountID (Reserved Word)

Disclaimer

Gets the account that has been selected in the Format Strategy dialog. The return value from this function can be used as an input to the other TradeManager functions.

Usage:

GetAccountID

Return:

Returns a string representation of the account associated with the strategy. Will return an empty string if called from an indicator.

Inputs:

None

Example:

```
vars: str("");
str = GetAccountID();
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetAccountLotSize (Reserved Word)

Disclaimer

Retrieve the size of 1 lot from the TradeManager’s Balances tab for a given forex account.

Usage:

```
GetAccountLotSize(Account)
```

Return:

Return an integer value identifying the number of currency units in a lot for a users account.. These values will be either 10,000 or 100,000 depending on the account. If the account is not valid or not a forex account, a 0 will be returned.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetAccountLotSize(GetAccountID);
```

GetAccountStatus (Reserved Word)

Disclaimer

Retrieve the Account Status from the TradeManager's Balances tab for the given equity or futures account.

Usage:

GetAccountStatus(Account)

Return:

Return an integer value identifying whether the account status is Closing Transaction Only, Liquidating Transaction Only, Restricted, Closed, Active, Inactive or Fed Margin Call.

The return value identifies the account status..

There a re also EasyLanguage constants to help identify and compare the account status return value,. these are defined as follows:

0 = asInvalid

1 = asActive

2 = asInactive

3 = asRestricted

4 = asClosed

5 = asClosingOnly

6 = asLiquidatingOnly

7 = asFedMarginCall

Inputs:

Account is the string representation of the account to check. GetAccountID returns the account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetAccountStatus(GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetAccountType (Reserved Word)

Disclaimer

Retrieve the Account Type from the TradeManager's Balances tab for the given equity or futures account.

Usage:

GetAccountType(Account)

Return:

Return an integer value identifying whether the account type is Future, Equity Cash, Equity Margin or Equity DVP.

The return value identifies the account type.

There a re also EasyLanguage constants to help identify and compare the account type return value,. these are defined as follows:

0 = atInvalid

1 = atCash

2 = atMargin

3 = atShort

4 = atDVP

5 = atFutures

Inputs:

Account is the string representation of the account to check. GetAccountID returns the account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetAccountType(GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetAllOrNone (Reserved Word)

Disclaimer

Identifies the current state of the “All or None” advanced order setting as set by SetAllOrNone.

Usage:

GetAllOrNone

Return:

Returns TRUE if “All or None” is currently selected otherwise returns FALSE.

Inputs:

None

Example:

```
Condition1 = GetAllOrNone;
```

GetAppInfo (Reserved Word)

Disclaimer

Returns a numeric value that refers to an attribute of the calling application based on a specified keyword.

Usage

```
GetAppInfo(InfoName);
```

Return

A numeric (double) value based on the keyword used (see **Return** column in table below).

Input

InfoName is a reserved word used to request information about a specific application attribute (see **InfoName** column in table below).

InfoName	Meaning	Return

`aiApplicationType`	Identifies the calling application	cUnknown = 0
		cChart = 1
		cRadarScreen = 2
		cOptionStation = 3

`aiBarSpacing`	Identifies the number of spaces displayed between bars on a chart.	Returns a positive non-zero value when called from a chart otherwise returns 0.
`aiHighestDispValue`	Identifies highest possible value of the underlying data visible on the screen for which the analysis technique is plotted. Note that this value is only useful if the chart is visible in the active workspace and the value is only updated when a new trade or barclose occurs.	Returns the highest displayed value when called from a chart otherwise returns 0.
`aiLeftDispDateTime`	Identifies date and time of the first (leftmost) bar displayed on the chart.	Returns a DateTime(double) value when called from a chart otherwise returns 0.
`aiLowestDispValue`	Identifies lowest possible value of the underlying data visible on the screen for which the analysis technique is plotted. Note that this value is only useful if the chart is visible in the active workspace and the value is only updated when a new trade or barclose occurs.	Returns the lowest displayed value when called from a chart otherwise returns 0.
`aiMacroEnabled`	Identifies whether the calling application can execute a macro.	Returns 1 if macros are enabled otherwise returns 0. Note that if the EasyLanguage code contains no macros 0 will be returned.
`aiMacroConf`	Identifies whether the calling application will	Returns 1 if macro confirmation is enabled

	display the macro confirmation dialog.	otherwise returns 0.

`aiOptionStationPane`	Identifies the OptionStation pane where the analysis technique is called from.	osInvalidAppType = 0
		osAssetPane= 1
		osOptionPane = 2
		osPositionPane = 3

`aiPLofAcctCurrency`	Identifies the state of the strategy properties "Base Currency" setting.	Returns 1 if the currency setting is Account and 0 if the currency setting is Symbol.
`aiRightDispDateTime`	Identifies date and time of the last (rightmost) bar displayed on the chart.	Returns a DateTime(double) value when called from a chart otherwise returns 0.
`aiRow`	Identifies the symbol's row number in RadarScreen.	Returns a positive non-zero value from a RadarScreen application else returns 0.
`aiSpaceToRight`	Identifies the number of bar spaces inserted to the right of the last bar on a chart.	Returns a non-zero value when called from a chart else returns 0.
`aiPercentChange`	Identifies whether the chart is displayed as a percent change chart.	Returns 1 if the chart is displayed using percent change else returns 0. If not called from a chart this will return 0.
`aiOptimizing`	Identifies whether the calling application is currently performing an optimization.	Returns 1 if the calling application is currently performing an optimization otherwise returns 0. This will return 0 when called from RadarScreen or OptionStation since currently these applications do not allow strategy testing.
`aiStrategyAuto`	Identifies whether the calling application is automating a strategy.	Returns 1 if the calling application is currently automating a strategy otherwise returns 0. This will return 0 when called from RadarScreen or OptionStation since currently these applications do not allow strategy testing.
`aiStrategyAutoConf`	Identifies whether the calling application is automating a strategy using order confirmation.	Returns 0 if the calling application is currently automating a strategy with order confirmation turned off otherwise returns 1 (even if no strategy is applied). This will return 0 when called from RadarScreen or OptionStation since currently these applications do not allow strategy testing.
`aiIntrabarOrder`	Identifies whether the calling application is	Returns 1 if the calling application is currently

	running a strategy that generates orders intra-bar.	running a strategy that generates orders intra-bar otherwise returns 0. This will return 0 when called from RadarScreen or OptionStation since currently these applications do not allow strategy testing.
aiAppId	A unique number used to identify the calling application.	Returns a non-zero integer.
aiRealTimeCalc	Identifies whether the calling application is calculating as a result of a real-time trades.	Returns 1 if the calculation results from a real-time trade otherwise 0 is returned.

Remarks

The value returned by GetAppInfo depends on the keyword specified. GetAppInfo can be called multiple times to return different application values..

Example

The following EasyLanguage instructions are used to check for RadarScreen as the calling application and then return the row of the calling symbol.:

```
Value1 = GetAppInfo(aiApplicationType);        // returns a value that identifies
the calling window type

if Value1 = 2 then  Value2 = GetAppInfo(aiRow);  // returns a positive row value
from RadarScreen
```

GetBackgroundColor (Reserved Word)

Disclaimer

Returns the numeric color value for the background color of the chart.

```
GetBackgroundColor
```

Remarks

The value returned is a number from 1 to 16 for the 16 possible colors.

Example

If you wanted to set a variable, `Value1`, to the color of the background, you could use the following syntax:

```
Value1 = GetBackgroundColor;
```

GetBDAccountEquity (Reserved Word)

Disclaimer

Retrieve the Beginning Day Account Equity amount from the TradeManager's Balances tab for the given futures account. Beginning Day Account Equity is defined as follows:

Beginning Day Cash + Beginning Day Trade Equity + Securities on Deposit

Usage:

GetBDAccountEquity(Account)

Return:

Returns a floating point value identifying the dollar amount of Beginning Day Account Equity for the given futures account. Returns zero for an equity or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetBDAccountEquity(GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetBDAccountNetWorth (Reserved Word)

Disclaimer

Retrieve the Beginning Day Account Net Worth amount from the TradeManager's Balances tab for the given equity account.

Usage:

GetBDAccountNetWorth (Account)

Return:

Returns a floating point value identifying the dollar amount of Beginning Day Account Net Worth for the given equity account. Returns zero for a futures or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetBDAccountNetWorth(GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetBDCashBalance (Reserved Word)

Disclaimer

Retrieve the Beginning Day Cash Balance amount from the TradeManager's Balances tab for the given futures account.

Usage:

GetBDCashBalance (Account)

Return:

Returns a floating point value identifying the dollar amount of Beginning Day Cash Balance for the given futures account. Returns zero for an equity or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetBDCashBalance (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetBDDayTradingBuyingPower (Reserved Word)

Disclaimer

Retrieve the Beginning Day DayTrading Buying Power amount from the TradeManager's Balances tab for the given equity account.

Usage:

GetBDDayTradingBuyingPower (Account)

Return:

Returns a floating point value identifying the dollar amount of Beginning Day DayTrading Buying Power for the given equity account. Returns zero for a futures or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetBDDayTradingBuyingPower (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetBDOvernightBuyingPower (Reserved Word)

Disclaimer

Retrieve the Beginning Day Overnight Buying Power amount from the TradeManager's Balances tab for the given equity account.

Usage:

GetBDOvernightBuyingPower (Account)

Return:

Returns a floating point value identifying the dollar amount of Beginning Day Overnight Buying Power for the given equity account. Returns zero for a futures or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetBDOvernightBuyingPower (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetBDTradeEquity (Reserved Word)

Disclaimer

Retrieve the Beginning Day TradeEquity amount from the TradeManager’s Balances tab for the given futures account. Beginning Day Trade Equity is defined as follows:

Sum (Long: (PrevLast – Avg_Cost) * BPV * Qty; Short: (Avg_Cost – PrevLast) * BPV * Qty)

(BPV = Big Point Value for a 1 point move in the security. (ES = $50.00, MSFT = $1.00))

Usage:

GetBDTradeEquity (Account)

Return:

Returns a floating point value identifying the dollar amount of Beginning Day Trade Equity for the given futures account. Returns zero for an equity or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetBDTradeEquity (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetBuyMinusSellPlus (Reserved Word)

Disclaimer

Identifies the current state of the “Buy on -/Sell on +” advanced order setting set by SetBuyMinusSellPlus

Usage:

GetBuyMinusSellPlus

Return:

Returns TRUE if “Buy on -/Sell on +” is currently selected otherwise returns FALSE.

Inputs:

None

Example:

```
  Condition1 = GetBuyMinusSellPlus;
```

GetBValue (Reserved Word)

Disclaimer

Returns an integer that represents a blue portion of the specified EasyLanguage RGB (16 million) color value.

```
GetBValue(BigRGBValue);
```

Where BigRGBValue is an EasyLangauge RGB color number from 0 to 16777215.

Remarks

The value returned is an integer value ranging from 0-255..

Example

To following returns a value of 56 for the blue portion of an EasyLangauge RGB color value of 3722240:

```
  Value1 = GetBValue(3722240);
```

GetCDRomDrive (Reserved Word)

Disclaimer

Returns a string expression of the drive letter for the first CD Rom drive detected.

```
GetCDRomDrive
```

Remarks

The value returned for a CD Rom drive of D would be "D".

Example

The following sets the variable Drive equal to the first CD Rom drive detected.

```
Variables: Drive("D");
Drive = GetCDRomDrive;
```

GetDiscretion (Reserved Word)

Disclaimer

Identifies the current state of the “Discretionary” advanced order setting as set by SetDiscretion

Usage:

GetDiscretion

Return:

A return value of zero indicates “Discretionary” is disabled. A return value greater that zero indicates the discretion price value used.

Inputs:

None

Example:

```
value1 = GetDiscretion;
```

GetExchangeName (Reserved Word)

Disclaimer

Returns the Name of the Exchange of the symbol to which the technique is applied.

`GetExchangeName`

Examples

`GetExchangeName` returns NYSE for IBM trading on the New York Stock Exchange

`GetExchangeName` returns CME for SP trading on the Chicago Mercantile Exchange

GetFundData (Reserved Word)

Disclaimer

Returns the numeric value of the specified fundamental data from some number of periods ago.

`GetFundData(sFieldName,nDataPointsBack)`

Where `sFieldName` is a text string containing the short name of the data to read. The `nDataPointsBack` parameter refers to the number of data periods ago from which to read the data.

Note All units, except share values, are converted to Millions for all periods. Percentages are returned as whole values (i.e. a value of 26.5 represents 26.5% and not 2650.00%)

Fundamental data names are listed in tables under the following categories:

Snapshot, Balance Sheet Assets & Liabilities, Cash Flow, Income, Shareholder Equity

Remarks

`GetFundData` returns fundamental data starting on the next business day after the post date reported by `GetFundPostDate`. For example, if a company announced earnings on a Friday, the first associated earnings data would be plotted on a chart as of the opening bar on the following Monday which is the next business day for trading.

Example

Assigns `Value1` the numeric value of fundamental data "ONET" (Net Income) from one period ago. Also, you can use the `GetLastFundDataError` reserved word to test the status of the previous 'Get' data call, where a non-zero value indicates that there was a data error.

```
Value1 = GetFundData("ONET", 1);

if GetLastFundDataError = fdrDataUnavailable then Print("Data Unavailable");
```

See Also

GetFundDataAsString, GetFundDataAsBoolean, GetFundPostDate, GetLastFundDataError

GetFundDataAsBoolean (Reserved Word)

Disclaimer

Returns the boolean (true/false) value of the specified fundamental data from some number of periods ago.

`GetFundDataAsBoolean(sFieldName,nDataPointsBack)`

Where `sFieldName` is a text string containing the short name of the data to read. The `nDataPointsBack` parameter refers to the number of data periods ago from which to read the data.

Note All units, except share values, are converted to Millions for all periods. Percentages are returned as whole values (i.e. a value of 26.5 represents 26.5% and not 2650.00%)

Fundamental data names are listed in tables under the following categories:

Snapshot, Balance Sheet Assets & Liabilities, Cash Flow, Income, Shareholder Equity

Remarks

`GetFundDataAsBoolean` returns fundamental data starting on the next business day after the post date reported by `GetFundPostDate`. For example, if a company announced earnings on a Friday, the first associated earnings data would be plotted on a chart as of the opening bar on the following Monday which is the next business day for trading.

Example

Assigns `Value1` the boolean value of fundamental data "Option" (Optionable stock) from two periods ago. Also, you can use the `GetLastFundDataError` reserved word to test the status of the previous 'Get' data call, where a non-zero value indicates that there was a data error.

```
Value1 = GetFundDataAsBoolean("Option", 2);
if GetLastFundDataError = fdrDataUnavailable then Print("Data Unavailable");
```

See Also

GetFundData, GetFundDataAsString, GetFundPostDate, GetLastFundDataError

GetFundDataAsString (Reserved Word)

Disclaimer

Returns the text string value of the specified fundamental data from some number of periods ago.

`GetFundDataAsString(sFieldName,nDataPointsBack)`

Where `sFieldName` is a text string containing the short name of the data to read. The `nDataPointsBack` parameter refers to the number of data periods ago from which to read the data.

Note All units, except share values, are converted to Millions for all periods. Percentages are returned as whole values (i.e. a value of 26.5 represents 26.5% and not 2650.00%)

Fundamental data names are listed in tables under the following categories:

Snapshot, Balance Sheet Assets & Liabilities, Cash Flow, Income, Shareholder Equity

Remarks

`GetFundDataAsString` returns fundamental data starting on the next business day after the post date reported by `GetFundPostDate`. For example, if a company announced earnings on a Friday, the first associated earnings data would be plotted on a chart as of the opening bar on the following Monday which is the next business day for trading.

Example

Assigns `Value1` the string value of fundamental data F_MGIND ('Industry Name') from two periods ago. Also, you can use the `GetLastFundDataError` reserved word to test the status of the previous 'Get' data call, where a non-zero value indicates that there was a data error.

```
Value1 = GetFundDataAsString("F_MGIND", 2);
if GetLastFundDataError = fdrDataUnavailable then Print("Data Unavailable");
```

See Also

GetFundData, GetFundDataAsBoolean, GetFundPostDate, GetLastFundDataError

GetFundPeriodEndDate (Reserved Word)

Disclaimer

Returns the end date of the reporting period for the specified fundamental data from some number of periods ago.

`GetFundPeriodEndDate(sFieldName,nDataPointsBack)`

Where `sFieldName` is a text string containing the short name of the data to read. The `nDataPointsBack` parameter refers to the number of data periods ago from which to read the data.

Fundamental data names are listed in tables under the following categories:

Snapshot, Balance Sheet Assets & Liabilities, Cash Flow, Income, Shareholder Equity

Remarks

`GetFundPeriodEndDate` returns the end date for the specified period of the referenced fundamental data field.

Example

Assigns `Value1` the period end date value of fundamental data "ONET" (Net Income) from one period ago. Also, you can use the `GetLastFundDataError` reserved word to test the status of the previous 'Get' data call, where a non-zero value indicates that there was a data error.

```
Value1 = GetFundPeriodEndDate("ONET", 1);
if GetLastFundDataError = fdrDataUnavailable then Print("Data Unavailable");
```

See Also

GetFundData, GetFundDataAsString, GetFundDataAsBoolean, GetFundPostDate, GetLastFundDataError

GetFundPostDate (Reserved Word)

Disclaimer

Returns the post date of the specified fundamental data from some number of periods ago.

`GetFundPostDate(sFieldName,nDataPointsBack)`

Where `sFieldName` is a text string containing the short name of the data to read. The `nDataPointsBack` parameter refers to the number of data periods ago from which to read the data.

Fundamental data names are listed in tables under the following categories:

Snapshot, Balance Sheet Assets & Liabilities, Cash Flow, Income, Shareholder Equity

Remarks

`GetFundPostDate` returns the actual date that a company announced earnings for the specified period while the first appearance of the new fundamental data based on those earnings will be on the next business day after the post date. For example, if a company announced earnings on a Friday, the first associated earnings data would be plotted on a chart as of the opening bar on the following Monday which is the next business day for trading.

Example

Assigns `Value1` the date value of fundamental data "ONET" (Net Income) from one period ago. Also, you can use the `GetLastFundDataError` reserved word to test the status of the previous 'Get' data call, where a non-zero value indicates that there was a data error.

```
Value1 = GetFundPostDate("ONET", 1);
if GetLastFundDataError = fdrDataUnavailable then Print("Data Unavailable");
```

See Also

GetFundData, GetFundDataAsString, GetFundDataAsBoolean, GetLastFundDataError

GetGValue (Reserved Word)

Disclaimer

Returns an integer that represents a green portion of the specified EasyLangauge RGB (16 million) color value.

```
GetGValue(BigRGBValue);
```

Where BigRGBValue is an EasyLangauge RGB color number from 0 to 16777215.

Remarks

The value returned is an integer value ranging from 0-255..

Example

To following returns a value of 204 for the green portion of an EasyLangauge RGB color value of 3722240:

```
  Value1 = GetGValue(3722240);
```

GetLastFundDataError (Reserved Word)

Disclaimer

Returns a numeric error code that identifies the status of the last referenced fundamental data.

```
GetLastFundDataError
```

Each numeric code has a descriptive EasyLanguage constant name to help identify and compare the type of error; these are defined as follows:

Constant Word	Value	Description
fdrOK	0	No error in last reference to fundamental data.
fdrInvalidField	1	Data name is not recognized.
fdrDataUnavailable	2	No data available for last referenced fundamental field.
fdrTypeMismatch	3	Data type of field didn't match reserved word used to reference value.
fdrFutureReference	4	A negative number is not allowed for the data points parameter.
fdrNoSnapshotHistory	5	Referencing historical data not allowed with snapshot field.
fdrNoMeaningfulValue	6	The value for the field has No Meaningful Figure (NMF).
fdrValueNotAvailable	7	The value for the field is Not Available (NA).

Example

Tests the error status from the previous 'Get' field call and sends a message to the print log indicating that the fundamental data for "ONET" (Net Income) is not available.

```
Value1 = GetFundData("ONET",1);
if GetLastFundDataError = fdrDataUnavailable then Print("Data Unavailable");
```

See Also

GetFundData, GetFundDataAsString, GetFundDataAsBoolean, GetFundPostDate

GetNonDisplay (Reserved Word)

Disclaimer

Identifies the current state of the “Non-Display” advanced order setting as set by SetNonDisplay.

Usage:

GetNonDisplay

Return:

Returns TRUE if “Non-Display” is currently selected otherwise returns FALSE.

Inputs:

None

Example:

```
Condition1 = GetNonDisplay;
```

GetNOW (Reserved Word)

Disclaimer

Identifies the current state of the “NOW (ECN’s Only)” advanced order setting set by SetNOW.

Usage:

GetNOW

Return:

Returns TRUE if “NOW (ECN’s Only)” is currently enabled otherwise returns FALSE..

Inputs:

None

Example:

```
   Condition1 = GetNOW;
```

GetNumAccounts (Reserved Word)

Disclaimer

Get the number of available equity and futures accounts. This number is used to enumerate the available accounts using the GetAccount() function.

Usage:

GetNumAccounts

Return:

Returns the number of available accounts.

Inputs:

None

Example:

```
   Value1 = GetNumAccounts;
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetNumPositions (Reserved Word)

Disclaimer

Get the number of open positions for the given equity or futures account. This number is used to enumerate the positions available accounts using the GetPositionSymbol function.

Usage:

GetNumPositions (Account)

Return:

Returns the number of open positions for the given account. Returns zero for an invalid account or if flat.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the account parameter needed here, or you can specify it manually.

Example:

```
  Value1 = GetNumPositions (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetOpenOrderInitialMargin (Reserved Word)

Disclaimer

Retrieve the Open Order Initial Margin amount from the TradeManager's Balances tab for the given futures account. Open Order Initial Margin is defined as follows:

Sum (Open Order Initial Margins of all positions in the given account)

Usage:

GetOpenOrderInitialMargin (Account)

Return:

Returns a floating point value identifying the dollar amount of Open Order Initial Margin for the given futures account. Returns zero for an equity or invalid account..

Inputs:

Account is the string representation of the account to check. GetAccountID returns the account parameter needed here, or you can specify it manually.

Example:

```
   Value1 = GetOpenOrderInitialMargin (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetPeg (Reserved Word)

Disclaimer

Identifies the current state of the “Peg” advanced order setting as set by SetPeg

Usage:

GetPeg

Return:

An Integer for the state of the Peg feature. A return value of PEGDisable indicates the “Peg” advanced order is disabled, otherwise the return value indicates the peg order is enabled and the peg type:

There a re also EasyLanguage constants to help identify and compare the Peg type return value, these are defined as follows:

0 = PegDisable (Peg Feature Disabled)

1 = PegBest (Enabled at: Best Bid for a Buy, Best Ask for a Sell Short)

2 = PegMid (Enabled at: Mid Price for both Buy and Sell Short)

Inputs:

None

Example:

```
Value1 = GetPeg;
```

GetPlotBGColor (Reserved Word)

Disclaimer

Returns the numeric color value of the plot (cell) background in a grid.

```
GetPlotBGColor(PlotNum);
```

Where `PlotNum` is a numeric expression representing a plot number.

Remarks

The value returned is an integer that represents the EasyLangauge RGB (16 million) color value from 0 to 16777215.

Note If `LegacyColorValue=True` then the value returned will be the nearest legacy color from 1 to 16. See LegacyColorValue for details.

Example

If you wanted to set a variable, `Value1`, to the cell color of the Plot1 background, you could use the following syntax:

```
   Value1 = GetPlotColor(1);
```

GetPlotColor (Reserved Word)

Disclaimer

Returns the numeric color value of a plot line in a chart.

```
GetPlotColor(PlotNum);
```

Where `PlotNum` is a numeric expression representing a plot number.

Remarks

The value returned is an integer that represents the EasyLangauge RGB (16 million) color value from 0 to 16777215.

Note If `LegacyColorValue=True` then the value returned will be the nearest legacy color from 1 to 16. See LegacyColorValue for details.

Example

If you wanted to set a variable, `Value1`, to the color of the Plot1 foreground, you could use the following syntax:

```
  Value1 = GetPlotColor(1);
```

GetPlotWidth (Reserved Word)

Disclaimer

Returns the numeric width value of a plot line in a chart.

```
GetPlotWidth(PlotNum);
```

Where `PlotNum` is a numeric expression representing a plot number.

Remarks

The value returned is a number representing the assigned width of the specified plot line.

Example

If you wanted to set a variable, `Value1`, to the width of the Plot1 line on a chart, you could use the following syntax:

```
   Value1 = GetPlotWidth(1);
```

GetPositionAveragePrice (Reserved Word)

Disclaimer

Identifies the average price of the equity or forex position for the given symbol in the given account.

Usage

GetPositionAveragePrice (Symbol, Account)

Return

Returns a double identifying the average price. If no position exists or if the symbol or account is invalid the return value will be zero.

Inputs

Symbol is a string that identifies the symbol. GetSymbolName returns the Symbol of the chart as the parameter needed here.

Account is a string representation of the account to check for the position in. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example

```
Value1 = GetPositionAveragePrice (GetSymbolName, GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetPositionMarketValue (Reserved Word)

Disclaimer

Retrieves the future or equity position’s Market Value for the given symbol in the given account. For Forex, the Account Currency Value of the position is returned.

Usage

GetPositionMarketValue (Symbol, Account)

Return

Returns a float value representing the Market Value for futures/equities or the Account Currency Value for Forex. If no position exists or if the symbol or account is invalid the return value will be zero.

Inputs

Symbol is a string that identifies the symbol. GetSymbolName returns the Symbol of the chart as the parameter needed here.

Account is a string representation of the account to check for the position in. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example

```
Value1 = GetPositionMarketValue (GetSymbolName, GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetPositionOpenPL (Reserved Word)

Disclaimer

Retrieves the future, equity, or Forex position’s Open P/L for the given symbol in the given account.

Usage

GetPositionOpenPL (Symbol, Account)

Return

Returns a floating point value identifying the position’s Open P/L. If no position exists or if the symbol or account is invalid the return value will be zero.

Inputs

Symbol is a string that identifies the symbol. GetSymbolName returns the Symbol of the chart as the parameter needed here.

Account is a string representation of the account to check for the position in. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example

```
Value1 = GetPositionOpenPL (GetSymbolName, GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetPositionQuantity (Reserved Word)

Disclaimer

Identifies the net quantity and side of the equity or futures position for the given symbol in the given account. For Forex, the number of lots is returned.

Usage

GetPositionQuantity (Symbol, Account)

Return

Returns an integer identifying the net position quantity for the given symbol and account. A negative value indicates the position is net short. A positive value indicates the position is net long. Zero indicates the position is net flat or the symbol or account is invalid.

Input:

Symbol is a string that identifies the symbol. GetSymbolName returns the Symbol of the chart as the parameter needed here.

Account is a string representation of the account to check for the position in. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example

```
  Value1 = GetPositionQuantity (GetSymbolName, GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetPositionSymbol (Reserved Word)

Disclaimer

Identifies the symbol associated with the Position location number in an alphabetically sorted list of account positions maintained by the TradeManager . This function can be used with GetNumPositions to enumerate available positions.

Note Adding or removing positions will change the sorted position list and may change the index value of existing positions.

Usage

GetPositionSymbol (strAccountNumber, iPositionId)

Return

Returns a string identifying the symbol associated with iPositionId location in a list of open positions. Returns an empty string for an invalid iPositionId.

Inputs

Account is a string representation of the account to check for the position in. GetAccountID returns the Account parameter needed here, or you can specify it manually.
iPositionId is an integer number with a value between 1 and the number returned by GetNumPositions.

Example

```
vars: str("");
str = GetPositionSymbol (GetAccountID ,1);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetPositionTotalCost (Reserved Word)

Disclaimer

Retrieves the future or equity position's Total Cost for the given symbol in the given account. No value is returned for Forex positions.

Usage

GetPositionTotalCost (Symbol, Account)

Return

Returns a float value representing the Total Cost. If no position exists or if the symbol or account is invalid the return value will be zero.

Inputs

Symbol is a string that identifies the symbol. GetSymbolName returns the Symbol of the chart as the parameter needed here.

Account is a string representation of the account to check for the position in. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example

```
Value1 = GetPositionTotalCost (GetSymbolName, GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetRoute (Reserved Word)

Disclaimer

Identifies the currently selected order routing set by SetRoute.

Usage

GetRoute

Return

Returns an integer value identifying the currently selected route.

These a re the EasyLanguage constants for routes and the values that are returned:

1 = rtIntelligent

2 = rtARCA

3 = rtBRUT

4 = rtBTRD

5 = rtINCA

6 = rtISLD

7 = rtSuperMont

8 = rtSuperDOT

Inputs

None

Example

```
  Value1 = GetRoute;
```

GetRTAccountEquity (Reserved Word)

Disclaimer

Retrieve the Real-time Account Equity amount from the TradeManager’s Balances tab for the given futures account. Real-time Account Equity is defined as follows:

Real-time Cash + Real-time Trade Equity + Securities on Deposit

Usage:

GetRTAccountEquity (Account)

Return:

Returns a floating point value identifying the dollar amount of Real-time Account Equity for the given futures account. Returns zero for an equity or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example:

```
   Value1 = GetRTAccountEquity (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetRTAccountNetWorth (Reserved Word)

Disclaimer

Retrieve the Real-time Account Net Worth amount from the TradeManager's Balances tab for the given equity account.

Usage:

GetRTAccountNetWorth (Account)

Return:

Returns a floating point value identifying the dollar amount of Real-time Account Net Worth for the given equity account. Returns zero for a futures or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetRTAccountNetWorth (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetRTCashBalance (Reserved Word)

Disclaimer

Retrieve the Real-time Cash Balance amount from the TradeManager's Balances tab for the given futures account. Real-time Cash Balance is defined as follows:

Beginning Day Cash - Commissions

Usage:

GetRTCashBalance (Account)

Return:

Returns a floating point value identifying the dollar amount of Real-time Cash Balance for the given futures account. Returns zero for an equity or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example:

```
   Value1 = GetRTCashBalance (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetRTCostOfPositions (Reserved Word)

Disclaimer

Retrieve the Real-time Cost of Positions amount from the TradeManager's Balances tab for the given equity account..

Usage:

GetRTCostOfPositions (Account)

Return:

Returns a floating point value identifying the dollar amount of Real-time Cost of Positions for the given equity account. Returns zero for a futures or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetRTCostOfPositions (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetRTDaytradingBuyingPower (Reserved Word)

Disclaimer

Retrieve the Real-time Daytrading Buying Power amount from the TradeManager's Balances tab for the given equity account.

Usage:

GetRTDaytradingBuyingPower (Account)

Return:

Returns a floating point value identifying the dollar amount of Real-time Daytrading Buying Power for the given equity account. Returns zero for a futures or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetRTDaytradingBuyingPower (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetRTInitialMargin (Reserved Word)

Disclaimer

Retrieve the Real-time Initial Margin amount from the TradeManager's Balances tab for the given futures account. Real-time Initial Margin is defined as follows:

Sum (Initial Margins of all positions in the given account)

Usage:

GetRTInitialMargin (Account)

Return:

Returns a floating point value identifying the dollar amount of Real-time Initial Margin for the given futures account. Returns zero for an equity or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetRTInitialMargin (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetRTMaintMargin (Reserved Word)

Disclaimer

Retrieve the Real-time Maintenance Margin amount from the TradeManager's Balances tab for the given futures account. Real-time Maintenance Margin is defined as follows:

Sum (Maintenance Margins of all positions in the given account)t)

Usage:

GetRTMaintMargin (Account)

Return:

Returns a floating point value identifying the dollar amount of Real-time Maintenance Margin for the given futures account. Returns zero for an equity or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetRTMaintMargin (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetRTOvernightBuyingPower (Reserved Word)

Disclaimer

Retrieve the Real-time Overnight Buying Power amount from the TradeManager's Balances tab for the given equity account.

Usage:

GetRTOvernightBuyingPower (Account)

Return:

Returns a floating point value identifying the dollar amount of Real-time Overnight Buying Power for the given equity account. Returns zero for a futures or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetRTOvernightBuyingPower (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetRTPurchasingPower (Reserved Word)

Disclaimer

RetrieRetrieve the Real-time Purchasing Power amount from the TradeManager’s Balances tab for the given futures account. Real-time Purchasing Power is defined as follows:

Real-time Cash + Real-time Trade Equity + Securities on Deposit + Initial Margin - Open Order Initial Margin

Usage:

GetRTPurchasingPower (Account)

Return:

Returns a floating point value identifying the dollar amount of Real-time Purchasing Power for the given futures account. Returns zero for an equity or invalid account.t.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetRTPurchasingPower (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetRTRealizedPL (Reserved Word)

Disclaimer

Retrieve the Real-time Realized P/L amount from the TradeManager’s Balances tab for the given equity account.

Usage:

GetRTRealizedPL (Account)

Return:

Returns a floating point value identifying the dollar amount of Real-time Realized P/L for the given equity account. Returns zero for a futures or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetRTRealizedPL (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetRTTradeEquity (Reserved Word)

Disclaimer

Retrieve the Real-time Trade Equity amount from the TradeManager's Balances tab for the given futures account. Real-time Trade Equity is defined as follows:

Sum (Long: (Last - Avg_Cost) * BPV * Qty; Short: (Avg_Cost - Last) * BPV * QTY)

(BPV = Big Point Value for a 1 point move in the security. (ES = $50.00, MSFT = $1.00))

Usage:

GetRTTradeEquity (Account)

Return:

Returns a floating point value identifying the dollar amount of Real-time Trade Equity for the given futures account. Returns zero for an equity or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetRTTradeEquity (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetRTUnrealizedPL (reserved word)

Disclaimer

Retrieve the Real-time Unrealized P/L amount from the TradeManager's Balances tab for the given equity account.

Usage:

GetRTUnrealizedPL (Account)

Return:

Returns a floating point value identifying the dollar amount of Real-time Unrealized P/L for the given equity account. Returns zero for a futures or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetRTUnrealizedPL (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetRValue (Reserved Word)

Disclaimer

Returns an integer that represents a red portion of the specified EasyLangauge RGB (16 million) color value.

```
GetRValue(BigRGBValue);
```

Where BigRGBValue is an EasyLangauge RGB color number from 0 to 16777215.

Remarks

The value returned is an integer value ranging from 0-255..

Example

To following returns a value of 0 for the red portion of an EasyLangauge RGB color value of 3722240:

```
   Value1 = GetRValue(3722240);
```

GetScreenName (Reserved Word)

Disclaimer

Returns a string containing the user name used when logging on to the TradeStation Network.

GetShaveImprove (Reserved Word)

Disclaimer

Identifies the amount by which the strategy limit price will be adjusted by SetShaveImprove.

Usage:

GetShaveImprove

Return:

Returns a double value indicating the amount the strategy limit price will be adjusted by.

Inputs:

None

Example:

```
Value1 = GetShaveImprove;
```

GetShowOnly (Reserved Word)

Disclaimer

Identifies the current state of the “Show Only” advanced order setting as set by SetShowOnly

Usage:

GetShowOnly

Return:

A return value of zero indicates “Show Only” is disabled. A return value greater that zero indicates the number of shares that are shown.

Inputs:

None

Example:

```
value1 = GetShowOnly;
```

GetStrategyName (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

GetSubscriberOnly (Reserved Word)

Disclaimer

Identifies the current state of the “Subscriber Only” advanced order setting as set by SetSubscriberOnly

Usage:

GetSubscriberOnly

Return:

Returns TRUE is “Subscriber Only” is currently selected otherwise returns FALSE.

Inputs:

None

Example:

```
   Condition1 = GetSubscriberOnly;
```

GetSymbolName (Reserved Word)

Disclaimer

Returns the Name of the symbol to which the technique is applied.

```
GetSymbolName
```

Remarks

Return value is a string.

Returns the root, year, and contract month for futures symbols.

Examples

`GetSymbolName` returns IBM for International Business Machines trading on the New York Stock Exchange

GetSystemName (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

GetTodaysRTTradeEquity (Reserved Word)

Disclaimer

Retrieve Today's Real-time Trade Equity amount from the TradeManager's Balances tab for the given futures account. Today's Real-time Trade Equity is defined as follows:

Beginning Day Trade Equity – Real-time Trade Equity

Usage:

GetTodaysRTTradeEquity (Account)

Return:

Returns a floating point value identifying the dollar amount of Today's Real-time Trade Equity for the given futures account. Returns zero for an equity or invalid account.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example:

```
   Value1 = GetTodaysRTTradeEquity (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

GetUnclearedDeposits (Reserved Word)

Disclaimer

Retrieve the Uncleared Deposits amount from the TradeManager's Balances tab for the given equity or futures account.

Usage:

GetUnclearedDeposits (Account)

Return:

Returns a floating point value identifying the dollar amount of Uncleared Deposits. Return value is zero if Account if not valid.

Inputs:

Account is the string representation of the account to check. GetAccountID returns the Account parameter needed here, or you can specify it manually.

Example:

```
Value1 = GetUnclearedDeposits (GetAccountID);
```

Note This reserved word uses processor intensive TradeManager calls that may adversely impact the performance of TradeStation and other applications on your computer.

Gr_Rate_P_EPS (Reserved Word)

Disclaimer

Retained for backward compatibility.

GradientColor (Reserved Word)

Disclaimer

This word returns a specific color from a user defined gradient color range, such as Blue to White.

Usage:

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

Return:

Returns a specific color from a user defined gradient color range, such as Blue to White

Inputs:

- Value = value being passed-in that is within the specified Min/Max values
- Min = Starting value for the gradient, where the StartColor is displayed
- Max = Ending value for the gradient, where the EndColor is displayed
- StartColor = Starting color of the gradient
- EndColor = Ending color of the gradient

Example:

```
Vars: RSIValue( 0 );
RSIValue = RSI( Close, 14 );
Plot1( RSIValue, "RSI");
SetPlotColor( 1, GradientColor( RSIValue, 20, 80, White, Blue));
```

Green (Reserved Word)

Disclaimer

Sets the plot color or background color to Green.

```
Green
```

Remarks

There are 16 colors available in EasyLanguage. For more information, see EasyLanguage Colors and Their Corresponding Numeric Values.

Examples

```
  Plot1(Value1, "Test", Green)
```

plots `Value1` with the name `Test`, and sets the color of `Plot1` to `Green`.

```
  TL_SetColor(1, Green)
```

sets the color of a TrendLine with a reference number of 1 to `Green`.

GrossLoss (Reserved Word)

Disclaimer

Returns the total dollar amount of all closed losing trades.

```
GrossLoss
```

Remarks

`GrossLoss` does not include the current open position, or the value of winning trades

Examples

`GrossLoss` returns -1000 if there are three losing trades of –500, -200, and -300.

`GrossLoss` returns 0 if there are no closed losing trades, even if the current position is losing.

GrossProfit (Reserved Word)

Disclaimer

Returns the total dollar amount of all closed winning trades.

```
GrossProft
```

Remarks

`GrossProft` does not include the current open position, or the value of losing trades

Examples

`GrossProft` returns 1000 if there are three winning trades of 500, 200, and 300.

`GrossProfit` returns 0 if there are no closed winning trades, even if the current position is winning.

H (Reserved Word)

Disclaimer

Reserved word used to return the high price of the specified time increment or group of ticks.

```
H
```

Remarks

Any time the High of a bar is needed, the letter `H` can be used as an equivalent.

Examples

```
H of 1 bar ago
```

returns the High price of the previous bar.

```
Average(H, 10)
```

returns the Average of the last 10 High prices.

Additional Example

You can check that the last two bars have High prices higher than the previous bar, the following language can be used in a ShowMe study:

```
   If H > H[1] and H[1] > H[2] then
    Plot1(H, "Rising");
```

High (Reserved Word)

Disclaimer

Reserved word used to return the high price of the specified time increment or group of ticks.

```
High
```

Remarks

The letter `H` can be used as an equivalent to `High`.

Examples

```
High of 1 bar ago
```

returns the High price of the previous bar.

```
Average(High, 10)
```

returns the Average of the last 10 High prices.

Additional Example

To check that the last two bars have High prices higher than the previous bar, the following language can be used in a ShowMe study:

```
   If High > High[1] and High[1] > High[2] then
    Plot1(High, "Rising");
```

High52Wk (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

Higher (Reserved Word)

Disclaimer

Synonym for stop or limit orders depending on the context used within a strategy.

Higher means Stop when used in the following context:

Example

```
Buy next bar at MyEntryPrice or Higher;
BuyToCover next bar at MyExitPrice or Higher;
```

Higher means Limit when used in the following context:

Example

```
SellShort next bar at MyEntryPrice or higher;
Sell next bar at MyEntryPrice or higher;
```

HistFundExists (Reserved Word)

Disclaimer

Reserved for backward compatibility.

HoursFromDateTime (Reserved Word)

Disclaimer

Returns the numeric value of hours for the specified DateTime.

```
HoursFromDateTime(dDateTime);
```

Where `dDateTime` is a double-precision decimal expression representing the Julian Date (since 1900) and time (since midnight).

Remarks

If a specific date is entered for the numeric expression, it must be a valid Julian date between January 1, 1900 and February 28, 2150, expressed in DateTime format.

Examples

`Value1 = HoursFromDateTime(37564.61200);` returns an hour value of 14 (military time) for the specified DateTime of 37564.61200 (11/4/2002 2:41:17 PM).

I (Reserved Word)

Disclaimer

Shortcut notation that returns the Open Interest of a bar, the number of contracts outstanding at the close of the bar.

```
I
```

Remarks

Anytime the Open Interest of a bar is needed, the letter I can be used in an equivalent fashion.

Examples

```
I of 1 bar ago
```

returns the Open Interest of the previous bar.

```
Average(I, 10)
```

returns the Average of the last 10 Open Interest values.

Additional Example

To check that the last two bars have Open Interest values higher than the previous bar, the following language can be used in a ShowMe study:

```
If I > I[1] and I[1] > I[2] then
 Plot1(High, "Rising");
```

I_AvgEntryPrice (Reserved Word)

Disclaimer

Returns the Average entry price of each open entry in a pyramided position.

`I_AvgEntryPrice`

Remarks

- `I_AvgEntryPrice` only returns the average entry price for open trades.
- `I_AvgEntryPrice` can only be used in a study.
- `I_AvgEntryPrice` will only return a value if a strategy is applied to the same data.

Examples

`I_AvgEntryPrice` returns 150 if three trades are currently open and were entered at a price of 130, 145, and 175.

`I_AvgEntryPrice` returns 50 if four trades are currently open and were entered at a price of 42, 53, 37, and 68.

I_ClosedEquity (Reserved Word)

Disclaimer

Returns the closed equity for a position which is made up of the strategy's total net profit when a position has been closed.

`I_ClosedEquity`

Remarks

- `I_ClosedEquity` only returns the profit or loss for closed trades.
- `I_ClosedEquity` can only be used in a study.
- `I_ClosedEquity` will only return a value if a strategy is applied to the same data.

I_CurrentContracts (Reserved Word)

Disclaimer

Returns the number of contracts held in all open positions.

```
I_CurrentContracts
```

Remarks

`I_CurrentContracts` can only be used in a study.

`I_CurrentContracts` will only return a value if a strategy is applied to the same data.

Examples

`I_CurrentContracts` returns 3 if there are three open positions of one contract each.

`I_CurrentContracts` returns 10 if there are three open positions of 3, 4, and 3 contracts respectively.

I_CurrentShares (Reserved Word)

Disclaimer

Returns the number of shares held in all open positions.

```
I_CurrentShares
```

Remarks

`I_CurrentShares` can only be used in a study.

`I_CurrentShares` will only return a value if a strategy is applied to the same data.

Examples

`I_CurrentShares` returns 300 if there are three open positions of one hundred shares each.

`I_CurrentShares` returns 1000 if there are three open positions of 300, 400, and 300 shares respectively.

I_MarketPosition (Reserved Word)

Disclaimer

Returns the current Market Position.

`I_MarketPosition`

Remarks

- `I_MarketPosition` returns the following numbers:

1 For a Long position
-1 For a Short position
0 For a Flat position

- `I_MarketPosition` can only be used in a study.
- `I_MarketPosition` will only return a value if a strategy is applied to the same data.

Examples

`I_MarketPosition` returns 1 if the current position is a Long position.

`I_MarketPosition` returns -1 if the current position is a Short position.

I_OpenEquity (Reserved Word)

Disclaimer

Returns the open equity for a position which is made up of the strategy's total net profit + open position profit/loss.

`I_OpenEquity`

Remarks

- `I_OpenEquity` only returns the profit or loss for open strategy positions.
- `I_OpenEquity` can only be used in a study.
- `I_OpenEquity` will only return a value if a strategy is applied to the same data.

If (Reserved Word)

Disclaimer

This word is used to introduce a condition that will be evaluated to determine execution of additional code.

`If`

Remarks

If can only be used to begin an If… Then or If… Then… Else statement.

In order to use an If… Then… Else statement, a Begin… End statement must follow Then as in the second example below.

Examples

```
If Condition1 Then
 {Your Code Line1}
```

If is used here to start the If… Then statement. The Line1 code will be executed if Condition1 returns a value of True. If Condition1 is false, the Line1 code will not be executed.

```
If Condition1 And Condition2 Then Begin
 {Your Code Line1}
 {Your Code Line2, etc.}
End
Else Begin
 {Your Code Line3}
 {Your Code Line4, etc.}
End;
```

If is used here to start the If… Then… Else statement. The Line1 and Line2 code will be executed if Condition1 and Condition2 return a value of True. If Condition1 or Condition2 is false, the Line3 and Line4 code will be executed.

IncludeSignal (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

IncludeSystem (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word IncludeSignal.

InfiniteLoopDetect (Reserved Word)

Disclaimer

The InfiniteLoopDetect keyword directive allows you to enable or disable infinite loop detection in an EasyLanguage document. Infinite loop detection logic will always be enabled by default (i.e. if no attribute is present detection logic will be enabled). If the same attribute appears more that once in a document, an error message will appear during verify.

Usage Example:

```
    [InfiniteLoopDetect = FALSE]  //  Turns off infinite loop detection
```

Note that attributes are local to the document (function, indicator, strategy, etc.) in which they are used so it is possible to have a function with detection disabled but the calling analysis technique with the logic enabled. Also, attributes are evaluated at compile time only so their values cannot be changed at run-time.

INITIALMARGIN (Reserved Word)

Disclaimer

INITIALMARGIN is an OptionStation reserved word that returns the initial margin of a position.

Input (Reserved Word)

Disclaimer

Before any name can be used as a user-defined input, the name to be used must be declared as an input. To declare an input, the reserved word `Input` is used.

```
 Input: Name(Value) ;
```

Where `Name` is the name of the input to be used, and `Value` is either a Numeric, True/False or String value used as the default value of the input.

Remarks

An input name can contain up to 20 alphanumeric characters plus the period (.) and the underscore (_).

An input name can not start with a number or a period (.).

Inputs are constants that can not start a statement.

The default value of an Input is declared by a number in parenthesis after the input name.

The use of the reserved words `Input`, and `Inputs` is identical.

Examples

```
Input: Length(10);
```

declares the constant `Length` initializes the value to ten.

```
Input: Price(Close);
```

declares the constant `Price` initializes the value to the `Close` of a bar.

Inputs (Reserved Word)

Disclaimer

Before any names can be used as user-defined inputs, the names to be used must be declared as inputs. To declare multiple inputs, the reserved word `Inputs` is used.

```
 Inputs: FirstName(FirstValue), NextName(NextValue) ;
```

Where `FirstName` and `NextName` are the names of the inputs to be used, and `FirstValue` and `NextValue` are Numeric, True/False or String values used as the default value of each input.

Remarks

An input name can contain up to 20 alphanumeric characters plus the period (.) and the underscore (_).

An input name can not start with a number or a period (.).

Inputs are constants that can not start a statement.

The default value of an Input is declared by a number in parenthesis after the input name.

The use of the reserved words `Input`, and `Inputs` is identical.

Examples

```
   Input: Price(Close), Length(10);
```

declares the constants `Price` and `Length`, initializing the values to the Close of the bar and ten, respectively.

```
   Input: SlowMA(9), FastMA(4);
```

declares the constants `SlowMA` and `FastMA`, initializing the values to nine and four, respectively.

InsideAsk (Reserved Word)

Disclaimer

Returns a numeric expression representing the current best ask for a symbol.

Note Does not reference history. In Chart Analysis,it will only plot a value from the current bar forward.

InsideBid (Reserved Word)

Disclaimer

Returns a numeric expression representing the current best bid for a symbol.

Note Does not reference history. In Chart Analysis,it will only plot a value from the current bar forward.

Inst_Percent_Held (Reserved Word)

Disclaimer

Retained for backward compatibility.

InStr (Reserved Word)

Disclaimer

Returns the location of String2 within String1.

```
        InStr(String1, String2) ;
```

Where `String1` represents the string that will be evaluated and `String2` represents the string to be located.

Remarks

- Returns the location of `String2`within `String1`, represented as the number of characters from the left side.
- If the specified string (`String2`) is found more than once in `String1`, the value returned refers to only the first occurrence.
- 0 is returned if the specified string (`String2`) does not exist in the evaluated string (`String1`).

Examples

```
  InStr("Net Profit in December", "Profit");
```

Returns the value 5, indicating that the string "Profit" begins at position 5 of `String1`.

```
  InStr("Net Profit in December", "January")
```

Returns the value 0 since the string "January" does not exist in `String1`.

INT (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

IntPortion (Reserved Word)

Disclaimer

Returns the integer portion of the specified number.

`IntPortion(Num) ;`

Where `Num` is a numeric expression representing the number for which you want the integer portion.

Examples

`IntPortion(4.5)` returns a value of 4.

`IntPortion(-1.72)` returns a value of -1.

IntraBarOrderGeneration (Reserved Word)

Disclaimer

The EasyLanguage attribute IntraBarOrderGeneration allows you to turn on or off the intrabar order generation flag. If the attribute is not present in any of the strategy code, users will be allowed to control the setting through the Calculations tab. If the attribute is present and set to TRUE, the ‘Enable intrabar order generation and calculation’ flag will be checked, the checkbox will be disabled, and the radio buttons will be available. If the attribute is present and set to FALSE, the ‘Enable intrabar order generation and calculation’ flag will unchecked and the checkbox and radio buttons will be disabled.

The syntax is as follows:

```
[IntrabarOrderGeneration = Value]   // where Value can be TRUE or FALSE
```

Note The use of square brackets is required for this reserved word.

Remarks

This attribute may only be used in a strategy. If used in any other type of module the following error should appear at verify time and the focus should be set to the attribute: “Attribute IntrabarOrderGeneration is only available for use in strategies.”

The intrabar order generation feature is only supported for single data stream charts. If applied to a multiple data stream chart, the status of the strategy will be turned off.

Example

```
 [IntrabarOrderGeneration = FALSE]  //  Strategy will only generate orders at close
of bar
```

Note Attributes are evaluated at compile time only so their values cannot be changed at run-time.

IntrabarPersist (Reserved Word)

Disclaimer

A reserved word used when declaring a variable (or array) that indicates that the value of the variable (or array element) can be updated on every tick. By default, the value of a variable or an array element is only updated at the close of each bar. This word declares that the value of a variable or array element is to be updated intrabar.

Examples

Declares the variable `Count` to an initial value of zero and increments the value of `Count` every time a tick is received and at the close of each bar..

```
Variable: IntrabarPersist Count(0);
Count = Count + 1 ;
```

(Default behavior without IntrabarPersist) Declares the variable `Count` to an initial value of zero and increments the value of `Count` at the close of the each bar.

```
Variable: Count(0);
Count = Count + 1 ;
```

Is (Reserved Word)

Disclaimer

IS is a Skip Word that performs no function within your code. It is used within a statement only to make the EasyLanguage code more readable.

Remarks

Skip words in EasyLanguage are not necessary, however, they sometimes make it easier to understand the purpose of the code.

Syntax Coloring in the EasyLanguage PowerEditor is dark green for all skip words by default.

Example

```
If the Close is > 100 Then
      Alert;
```

The word "is" is not necessary and the code functions the same way regardless of its presence.

IsFundDataAvailable (Reserved Word)

Disclaimer

Returns true if data is available for the specified fundamental data from some number of periods ago. This doesn't return any data value, but does indicate that there is data to read.

```
IsFundDataAvailable(Name,DataPointsBack)
```

Where `Name` is a text string containing the short name of the data to read. The `DataPointsBack` parameter refers to the number of data periods ago from which to read the data.

Fundamental data names are listed in tables under the following categories:

Snapshot, Balance Sheet Assets & Liabilities, Cash Flow, Income, Shareholder Equity

Example

The conditional ACTION (your own EL statement(s)) will be executed if data is available for fundamental data "ONET" (Net Income) from two periods ago.

```
   if IsFundDataAvailable("ONET", 2) then ACTION;
```

See Also

IsValidFundField

IsValidFundField (Reserved Word)

Disclaimer

Returns true if the specified fundamental data name is valid.

`IsValidFundField(Name)`

Where `Name` is a text string containing the short name of the data to read. The `DataPointsBack` parameter refers to the number of data periods ago from which to read the data.

Fundamental data names are listed in tables under the following categories:

Snapshot, Balance Sheet Assets & Liabilities, Cash Flow, Income, Shareholder Equity

Example

The conditional ACTION (your own EL statement) will be executed if fundamental field "ONET" (Net Income) exists.

```
if IsValidFundField("ONET") then ACTION;
```

See Also

IsFundDataAvailable

IVolatility (Reserved Word)

Disclaimer

Returns the daily average weighted Implied Volatility of the options for an underlying stock, index, or future. Allows historical data (bars back) references.

Example

```
Value1 = IVolatility;
```

or

```
Value2 = IVolatility[2]     // of two bars ago
```

JulianToDate (Reserved Word)

Disclaimer

This reserved word returns a numeric expression representing the EasyLanguage date equivalent to the specified Julian Date.

```
JulianToDate(jDate);
```

Where `jDate` is a numeric expression representing the Julian date to convert into an EasyLanguage date (YYYMMDD format).

Examples

`JulianToDate(36011)` returns a value of 980804 for the date August 4, 1998.

`JulianToDate(36457)` returns a value of 991024 for the date October 24, 1999.

Additional Example

The following statement obtains the Julian Date of the day 20 calendar days ahead of the date of the current bar, and converts the result into an EasyLanguage date:

```
  Value1 = JulianToDate(DateToJulian(Date) + 20);
```

The expression inside parentheses (the reserved word `DateToJulian`) is evaluated first. It converts the date of the current bar to a Julian Date. Then, the number 20 is added to the resulting Julian Date. This Julian Date is then the parameter for the reserved word `JulianToDate`, which converts the Julian Date to an EasyLanguage date, in the format YYYMMDD. This EasyLanguage date is stored in the variable `Value1`.

L (Reserved Word)

Disclaimer

Shortcut notation that returns the Low of the bar.

```
L
```

Remarks

Anytime the Low of a bar is needed, the letter `L` can be used in an equivalent fashion.

Examples

```
L of 1 bar ago
```

returns the Low price of the previous bar.

```
Average(L, 10)
```

returns the Average of the last 10 Low prices.

Additional Example

To check that the last two bars have `Low` prices lower than the previous bar, the following language can be used in a ShowMe study:

```
If L < L[1] and L[1] < L[2] then
 Plot1(High, "Falling");
```

LargestLosTrade (Reserved Word)

Disclaimer

Returns the dollar value of the largest closed losing trade.

```
LargestLosTrade
```

Examples

`LargestLosTrade` returns -500 if there are three losing trades of –500, -200, and -300.

`LargestLosTrade` returns 0 if there are no closed losing trades, even if the current position is losing.

LargestWinTrade (Reserved Word)

Disclaimer

Returns the dollar value of the largest closed winning trade.

```
LargestWinTrade
```

Examples

`LargestWinTrade` returns 500 if there are three winning trades of 500, 200, and 300.

`LargestWinTrade` returns 0 if there are no closed winning trades, even if the current position is winning.

Last (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the price of the last completed trade.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

Last_Split_Date (Reserved Word)

Disclaimer

Retained for backward compatibility.

Last_Split_Fact (Reserved Word)

Disclaimer

Retained for backward compatibility.

LastCalcJDate (Reserved Word)

Disclaimer

Returns the Julian date for the last completed bar.

```
LastCalcJDate
```

Examples

`LastCalcJDate` returns a value of 36011 if the last bar was completed on August 4, 1998.

`LastCalcJDate` returns a value of 36173 if the last bar was completed on January 13, 1999.

LastCalcMMTime (Reserved Word)

Disclaimer

Returns the time, in number of minutes since midnight, for the last completed bar.

```
LastCalcMMTime
```

Examples

`LastCalcMMTime` returns a value of 540 if the last bar was completed at 9:00 am.

`LastCalcMMTime` returns a value of 835 if the last bar was completed at 1:55 pm.

LastTradingDate (Reserved Word)

Disclaimer

Refers to the last day an option, future, position leg, or asset may be traded.

```
LastTradingDate
```

Example

```
If DateToJulian( LastTradingDate ) - DateToJulian( Date ) < 5 then Alert("Less than
5 trading days left");
```

LEAPYear (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

LeftSide (Reserved Word)

Disclaimer

This reserved word is used with the other ActivityBar reserved words to specify the side of the ActivityBar you want to reference. It specifies that you are referencing the left side of the bar.

```
LeftSide
```

Example

The following statement obtains the number of cells on the left side of a bar, for the row corresponding to the closing price:

```
Value1 = AB_GetNumCells(Close of Data1, LeftSide);
```

LeftStr (Reserved Word)

Disclaimer

Returns the leftmost character(s) of a text string.

```
LeftStr(Str1, sSize);
```

Where `Str1` is the string expression to reduce. It must be enclosed in quotation marks.

`sSize` is the numeric expression indicating the number of characters that will be retained, counting from the beginning of the string expression.

Example

`LeftStr("Net Profit", 3)` returns the string expression "`Net`".

LEG (Reserved Word)

Disclaimer

LEG is an OptionStation reserved word, it is a modifier to other position leg related reserve words.

LegacyColorToRGB (Reserved Word)

Disclaimer

Returns an integer that represents the EasyLangauge RGB (16 million) color value from 0 to 16777215.

```
LegacyColorToRGB(LegacyColorValue);
```

Where LegacyColorValue is an EasyLangauge legacy color number from 1 to 16.

Remarks

The value returned is an integer value ranging from 1-16.

Example

To following returns an RGB value of 16711680 for the legacy color value 2 (blue):

```
  Value1 = LegacyColorToRGB(2);
```

LegacyColorValue (Reserved Word)

An EasyLanguage attribute that lets you specify what version of colors (legacy or RGB) to use when interpreting the code. Thus, in order to use the old (legacy) color value functionality, the user would need to specify this in his EL code, as shown in the example below:

```
[LegacyColorValue = True]
Plot1(Close) ;
SetPlotColor(1, Blue) ;
```

In the above example, Blue refers to the legacy color number for blue which is 2.

When ‘LegacyColorValue’ is TRUE, color names referenced in the EL code should return their legacy 16-bit color value. When ‘LegacyColorValue’ is FALSE or when ‘LegacyColorValue’ is not specified (default), color names specified in the EL code should return the 32-bit RGB color value.

This attribute can only be specified once within a specific EasyLanguage document, otherwise the code will not verify and an error message will appear.

Note The attribute `[LegacyColorValue = true]` is automatically inserted into imported EasyLanguage documents (created prior to v8.1) to ensure proper use of colors. To use RGB colors, simply remove this line or set the value to false.

LEGTYPE (Reserved Word)

Disclaimer

LEGTYPE is an OptionStation reserved word that returns type of position leg: asset, future, call, or put.

LightGray (Reserved Word)

Disclaimer

Sets the plot color or background color to Light Gray.

```
LightGray
```

Remarks

There are 16 colors available in EasyLanguage. For more information, see EasyLanguage Colors and Their Corresponding Numeric Values.

Examples

```
   Plot1(Value1, "Test", LightGray)
```

plots `Value1` with the name `Test`, and sets the color of `Plot1` to `LightGray`.

```
   TL_SetColor(1, LightGray)
```

sets the color of a TrendLine with a reference number of 1 to `LightGray`.

Limit (Reserved Word)

Disclaimer

Reserved word used in an Entry or Exit statement to specify how to fill an order.

```
Buy next bar at Value1 Limit;
```

Remarks

`Limit` orders can only be executed on the next bar.

`Limit` can be read as "this price or better", meaning lower for a Long Entry and Short Exit, higher for a Short Entry and Long Exit.

`Limit` orders require a reference price.

Examples

```
Buy next bar at 75 Limit;
```

Generates an order to enter a long position on the next bar at a price of 75 or lower.

```
SellShort next bar at 75 Limit;
```

Generates an order to enter a short position on the next bar at a price of 75 or higher.

Additional Example

```
Sell next bar at 75 Limit;
```

Generates an order to exit a long position on the next bar at a price of 75 or higher.

LiveDate (Reserved Word)

Disclaimer

Reserved for future use. LiveDate is the date that a Strategy went live.

LiveTime (Reserved Word)

Disclaimer

Reserved for future use. LiveTime is the time that a Strategy went live.

Log (Reserved Word)

Disclaimer

Returns the natural logarithm of a number.

`Log(Num)`

Where `Num` is a numeric expression to be used in the calculation.

Remarks

A logarithm is the exponent that indicates the power to which a number is raised to produce a given number. This reserved word uses a base e system.

Log can only be calculated for positive numbers.

Examples

`Log(4.5)` returns a value of 1.5041.

`Log(172)` returns a value of 5.1475.

LONG (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

LongRollAdj (Reserved Word)

Disclaimer

Reserved word that returns an end-of-day roll adjustment factor for long forex positions.

```
LongRollAdj
```

Remarks

LongRollAdj is used to apply a long position roll adjustment for p/l calculations on forex long positions that are held overnight to account for fluctuations in daily interest rates.

Examples

Assigns to Value1 the long position roll adjustment factor for the current bar.

```
Value1 = LongRollAdj;
```

Assigns to Value2 the long position roll adjustment factor from 5 bars ago.

```
Value2 = LongRollAdj[5]
```

LossCF (Reserved Word)

Disclaimer

Reserved word that returns a loss currency correction factor that may be used for calculating a loss when a trade is exited.

Remarks

`LossCF` is used to apply a currency conversion factor to loss calculations when a symbol reports its value in a currency that is different from that of the clients trading account.

See About Currency Conversion for information about currency conversion factors.

Low (Reserved Word)

Disclaimer

Represents the low price of the specified time increment or group of ticks.

```
Low
```

Remarks

`L` can be used in place of `Low`.

Examples

```
Low of 1 bar ago
```

returns the Low price of the previous bar

```
Average(Low, 10)
```

returns the Average of the last 10 Low prices

Additional Example

To check that the last two bars have Low prices lower than the previous bar, the following language can be used in a ShowMe study:

```
If Low < Low [1] AND Low [1] < Low [2] Then
 Plot1(Low, "Decline");
```

Low52Wk (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

Lower (Reserved Word)

Disclaimer

Synonym for stop or limit orders depending on context used within a strategy.

Lower means Limit when used in the following context:

Example

```
   Buy next bar at MyEntryPrice or Lower;
   BuyToCover next bar at MyExitPrice or Lower;
```

Lower means Stop when used in the following context:

Example

```
   SellShort next bar at MyEntryPrice or Lower;
   Sell next bar at MyEntryPrice or Lower;
```

LowerStr (Reserved Word)

Disclaimer

Used to convert a string expression to lowercase letters.

```
LowerStr("Str1") ;
```

Where `Str1` is the string expression to change to lowercase letters. Make sure the string expression is enclosed in quotation marks.

Example

`LowerStr("Net Profit")` returns the string expression "`net profit`".

LPBOOL (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

LPBYTE (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

LPDOUBLE (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

LPDWORD (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

LPFLOAT (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

LPINT (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

LPLONG (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

LPSTR (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

LPWORD (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

LTD (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the EasyLanguage date of the last trading date for the symbol. The last trading date is defined by the expiration rule chosen for the symbol.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

LTD (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the Julian date (accurate to the second) of the last trading date for the symbol. The last trading date is defined by the expiration rule chosen for the symbol.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

Magenta (Reserved Word)

Disclaimer

Sets the plot color or background color to Magenta.

```
Magenta
```

Remarks

There are 16 colors available in EasyLanguage. For more information, see EasyLanguage Colors and Their Corresponding Numeric Values.

Examples

```
   Plot1(Value1, "Test", Magenta)
```

plots `Value1` with the name `Test`, and sets the color of `Plot1` to `Magenta`.

```
   TL_SetColor(1, Magenta)
```

sets the color of a TrendLine with a reference number of 1 to `Magenta`.

MakeNewMovieRef (Reserved Word)

Disclaimer

This reserved word creates a new video clip and returns a numeric value representing the ID number of the new video clip created.

```
Value1 = MakeNewMovieRef;
```

`Value1` is any numeric variable or array.

Remarks

The reference number should be a whole positive number.

Notes

Once you create the video clip using this reserved word, you can add one or more .avi files to it using the reserved word `AddToMovieChain`. You must save the ID number of the video clip as it will be the way to reference the video clip in order to add .avi files as well as play it.

Example

The following statement creates a new video clip and assigns the ID number to the variable `Value1`:

```
Value1 = MakeNewMovieRef;
```

Margin (Reserved Word)

Disclaimer

Retained for backward compatibility.

Market (Reserved Word)

Disclaimer

Reserved word used to indicate the next trade or tick, not specific to the price.

```
Market
```

Remarks

Market is normally used in trading strategies.

Example

To enter a trade when the price of entry is not an issue:

```
Buy next bar at market;
```

MarketPosition (Reserved Word)

Disclaimer

Returns a numeric value for a short or long position for the specified position.

`MarketPosition(Num)`

Where `Num` is a numeric expression representing the number of positions ago.

Remarks

- -1 is returned for a short position, and 1 is returned for a long position. 0 is returned if the current position is specified, and you are not currently in the market.
- This function can only be used in the evaluation of strategies.

Example

`MarketPosition(0)` returns a value of 0 if the current position is flat.

MaxBarsBack (Reserved Word)

Disclaimer

All trading strategies, analysis techniques, and functions that refer to past data will need to wait a certain number of bars before they can start performing calculations. This waiting period can be adjusted for any analysis technique, and it is called *Maximum number of bars a study will reference*, or `MaxBarsBack`. For example, a 10-bar moving average would require a `MaxBarsBack` setting of 10 to begin calculating, which is 9 historical bars and the current bar.

```
MaxBarsBack
```

Remarks

The `MaxBarsBack` setting may prevent values from being displayed when an indicator is first applied. When enough data sets have been acquired to satisfy the `MaxBarsBack` setting, values will appear.

When analysis techniques are applied to more than one data set at a time (for example, a multi-data chart), each data set must meet the `MaxBarsBack` requirement individually before values will be generated.

MaxBarsForward (Reserved Word)

Disclaimer

Represents the number of bars to the right of the last bar on the chart.

Remarks

`MaxBarsForward` returns the number of bars used for space to the right.

MaxConsecLosers (Reserved Word)

Disclaimer

Represents the longest chain of consecutive closed-out losing trades.

```
MaxConsecLosers
```

Examples

`MaxConsecLosers` returns 5 if the largest number of consecutive closed losing trades is 5.

`MaxConsecLosers` returns 2 if the largest number of consecutive closed losing trades is 2.

MaxConsecWinners (Reserved Word)

Disclaimer

Represents the longest chain of consecutive closed-out winning trades.

```
MaxConsecWinners
```

Examples

`MaxConsecWinners` returns 5 if the largest number of consecutive closed winning trades is 5.

`MaxConsecWinners` returns 2 if the largest number of consecutive closed winning trades is 2.

MaxContractProfit (Reserved Word)

Disclaimer

Calculates the Maximum Position Profit per contract or share. `MaxContractProfit` will divide the complete position profit by the number of contracts/shares for the maximum position profit for a single share/contract

```
MaxContractProfit
```

Example

`Value1` will hold the maximum position profit for the current position.

```
Value1 = MaxContractProfit;
```

Additional Example

If you wanted to exit a long position after the profit per share reached $25, you could use the following syntax:

```
If MaxContractProfit >= 25 Then
 Sell This Bar on Close;
```

MaxContracts (Reserved Word)

Disclaimer

Returns the maximum number of contracts held during the specified position.

```
MaxContracts(Num)
```

Where `Num` is a numeric expression representing the number of positions ago (up to a maximum of ten).

Remarks

This function can only be used in the evaluation of strategies. It does not require an input, however, by using the input `Num`, you can obtain the specified value from a previous position, up to ten positions ago.

Example

`MaxContracts(2)` returns a value of 3 if the second most recent position held a maximum of 3 contracts.

MaxContractsHeld (Reserved Word)

Disclaimer

Returns the maximum number of contracts held at any one time.

```
MaxContractsHeld
```

Example

`MaxContractsHeld` returns a value of 3 if the maximum number of contracts held in any position was 3.

MaxEntries (Reserved Word)

Disclaimer

Returns the maximum number of entries of the specified position.

```
MaxEntries(Num)
```

Where `Num` is a numeric expression representing the number of positions ago (up to a maximum of ten).

Remarks

This function can only be used in the evaluation of strategies. It does not require an input, however, by using the input `Num`, you can obtain the specified value from a previous position, up to ten positions ago.

Example

`MaxEntries(2)` returns a value of 3 if the second most recent position consisted of 3 separate entries.

MAXGAIN (Reserved Word)

Disclaimer

MAXGAIN is an OptionStation reserved word that returns the theoretical maximum gain possible for a position.

MaxIDDrawDown (Reserved Word)

Disclaimer

Returns the largest drop in equity throughout the entire trading period.

```
MaxIDDrawDown
```

Remarks

`MaxIDDrawDown` returns the true number of dollars needed to sustain the largest equity dip.

Examples

`MaxIDDrawDown` returns 5000 if the largest equity dip throughout the entire trading period is 5000.

`MaxIDDrawDown` returns 2000 if the largest equity dip throughout the entire trading period is 2000.

MaxList (Reserved Word)

Disclaimer

Returns the highest value of the specified inputs.

`MaxList(Num1, Num2, Num3, etc.)`

Where `Num1, Num2` and `Num3` are numeric expressions representing values to be used in the calculation.

Examples

`MaxList(45, 72, 86, 125, 47)` returns a value of 125.

`MaxList(18, 67, 98, 24, 65, 19)` returns a value of 98.

MaxList2 (Reserved Word)

Disclaimer

Returns the second highest value of the specified the inputs.

`MaxList2(Num1, Num2, Num3, etc.)`

Where `Num1, Num2` and `Num3` are numeric expressions representing values to be used in the calculation.

Examples

`MaxList2(45, 72, 86, 125, 47)` returns a value of 86.

`MaxList2(18, 67, 98, 24, 65, 19)` returns a value of 67.

MAXLOSS (Reserved Word)

Disclaimer

MAXLOSS is an OptionStation reserved word that returns the theoretical maximum loss possible for a position.

MaxPositionLoss (Reserved Word)

Disclaimer

Returns the largest loss of the specified position.

`MaxPositionLoss(Num)`

Where `Num` is a numeric expression representing the number of positions ago (up to a maximum of ten).

Remarks

This function can only be used in the evaluation of strategies. It does not require an input, however, by using the input `Num`, you can obtain the specified value from a previous position, up to ten positions ago.

Example

`MaxPositionLoss(1)` returns a value of –3750.00 if the second most recent position had a maximum loss of 3750.00.

MaxPositionProfit (Reserved Word)

Disclaimer

Returns the largest gain of the specified position.

`MaxPositionProfit(Num)`

Where `Num` is a numeric expression representing the number of positions ago (up to a maximum of ten).

Remarks

This function can only be used in the evaluation of strategies. It does not require an input, however, by using the input `Num`, you can obtain the specified value from a previous position, up to ten positions ago.

Example

`MaxPositionProfit(1)` returns a value of 29200.00 if the second most recent position had a maximum profit of 29200.00.

MaxShares (Reserved Word)

Disclaimer

Returns the maximum number of shares held during the specified position.

`MaxShares(Num)`

Where `Num` is a numeric expression representing the number of positions ago (up to a maximum of ten).

Remarks

This function can only be used in the evaluation of strategies. It does not require an input, however, by using the input `Num`, you can obtain the specified value from a previous position, up to ten positions ago.

Example

`MaxShares(2)` returns a value of 300 if the second most recent position held a maximum of 300 shares.

MaxSharesHeld (Reserved Word)

Disclaimer

Returns the maximum number of shares held at any one time.

`MaxSharesHeld`

Example

`MaxSharesHeld` returns a value of 300 if the maximum number of contracts held in any position was 300.

MEDIUM (Reserved Word)

Disclaimer

Reserved for future use. FINE specifies a fine automatic optimization value for function inputs.

MessageLog (Reserved Word)

Disclaimer

Sends information to the Print Log.

```
MessageLog(Parameters) ;
```

Where `Parameters` may be a single string or numeric expression, or a multiple string or numeric expression, with each expression separated by a comma.

Examples

To send the date, time, and close information to the Print Log, you would use:

```
  MessageLog(Date, Time, Close);
```

MidStr (Reserved Word)

Disclaimer

Returns a string expression of `Siz` characters taken from `String` beginning with the `Pos` character measured from the left.

```
MidStr("String", Pos, Siz);
```

Where `String` is the string expression to reduce. It must be enclosed in quotation marks. `Pos` is the numeric expression indicating how many characters from the beginning of the string expression to remove and `Siz` is the numeric expression indicating the number of characters to be retained. Any additional characters to the right of these are removed.

Example

`MidStr("Net Profit in December", 5, 6)` returns the string expression "`Profit`".

MilliSecondsFromDateTime (Reserved Word)

Disclaimer

Reserved for future use.

MinList (Reserved Word)

Disclaimer

Returns the lowest value of the specified inputs.

`MinList(Num1, Num2, Num3, etc.)`

Where `Num1, Num2` and `Num3` are numeric expressions representing values to be used in the calculation.

Examples

`MinList(45, 72, 86, 125, 47)` returns a value of 45.

`MinList(18, 67, 98, 24, 65, 19)` returns a value of 18.

MinList2 (Reserved Word)

Disclaimer

Returns the second lowest value of the specified the inputs.

`MinList2(Num1, Num2, Num3, etc.)`

Where `Num1, Num2` and `Num3` are numeric expressions representing values to be used in the calculation.

Examples

`MinList2(45, 72, 86, 125, 47)` returns a value of 47.

`MinList2(18, 67, 98, 24, 65, 19)` returns a value of 19.

MinMove (Reserved Word)

Disclaimer

Returns a numeric expression (whole number) for calculating the minimum movement of the share or contract price for a symbol. MinMove is the numerator of the formula (MinMove / PriceScale) used to calculate the actual decimal value of the minimum price move. For example, the returned value of MinMove for the E-Mini S&P is 25, which results in minimum price move of .25 for an E-Mini S&P contract (MinMove divided by PriceScale of 100). Any stock would have a MinMove of 1, which results in a minimum price move of .01 for any stock (MinMove divided by PriceScale of 100).

```
MinMove
```

Examples

`MinMove / PriceScale` calculates the minimum price move for a share or contract.

`MinMove / PriceScale * BigPointValue` returns the profit or loss value associated with a minimum price move of a share or contract.

See Also

PriceScale

PointValue

BigPointValue

MinutesFromDateTime (Reserved Word)

Disclaimer

Returns the numeric value of minutes for the specified DateTime.

```
MinutesFromDateTime(dDateTime);
```

Where `dDateTime` is a double-precision decimal expression representing the Julian Date (since 1900) and time (since midnight).

Remarks

If a specific date is entered for the numeric expression, it must be a valid Julian date between January 1, 1900 and February 28, 2150, expressed in DateTime format.

Examples

`Value1 = MinutesFromDateTime(37564.61200);` returns a minutes value of 41 for the specified DateTime of 37564.61200 (11/4/2002 2:41:17 PM).

MIVONASK (Reserved Word)

Disclaimer

MIVONASK is an OptionStation reserved word that returns the market implied volatility of the current modeled ask price for an option.

MIVONBID (Reserved Word)

Disclaimer

MIVONBID is an OptionStation reserved word that returns the market implied volatility of the current modeled bid price for an option.

MIVONCLOSE (Reserved Word)

Disclaimer

MIVONCLOSE is an OptionStation reserved word that returns the market implied volatility of the current last trade price for an option.

MIVONRAWASK (Reserved Word)

Disclaimer

MIVONRAWASK is an OptionStation reserved word that returns the market implied volatility of the current best ask price for an option.

MIVONRAWBID (Reserved Word)

Disclaimer

MIVONRAWBID is an OptionStation reserved word that returns the market implied volatility of the current best bid price for an option.

Moc (Reserved Word)

Disclaimer

This word has been reserved for future use.

Mod (Reserved Word)

Disclaimer

Divides two numbers and returns the remainder.

```
Mod(Num, Divisor)
```

Where `Num` is a numeric expression to be used in the calculation and `Divisor` is a numeric expression representing the divisor.

Examples

`Mod(17, 5)` returns a value of 2.

`Mod(457, 9)` returns a value of 7.

MODELPOSITION (Reserved Word)

Disclaimer

Retained for backward compatibility.

MODELPRICE (Reserved Word)

Disclaimer

MODELPRICE is an OptionStation reserved word that returns the underlying asset price used as an input to a Pricing Model for a core calculation event.

MODELRATE (Reserved Word)

Disclaimer

MODELRATE is an OptionStation reserved word that returns the interest rate used as an input to a Pricing Model for a core calculation event.

MODELRATE2 (Reserved Word)

Disclaimer

Reserved for future use. MODELRATE2 is an OptionStation reserved word that returns a second interest rate used as an input to a Pricing Model for a core calculation event.

MODELVOLATILITY (Reserved Word)

Disclaimer

MODELVOLATILITY is an OptionStation reserved word that returns the volatility used as an input to a Pricing Model for a core calculation event

Monday (Reserved Word)

Disclaimer

Returns the number 1 as the value for Monday.

```
Monday
```

Example

`Monday` returns a value of 1 as the value for Monday.

MoneyMgtStopAmt (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word SetStopLoss.

Month (Reserved Word)

Disclaimer

Returns the corresponding month for the specified calendar date. (1 = Jan, 12 = Dec)

```
Month(cDate)
```

Where `cDate` is a numeric expression representing the six or seven digit calendar date in the format YYMMDD or YYYMMDD respectively. (1999 = 99, 2001 = 101)

Examples

`Month(990613)` returns a value of 6 because 990613 represents June 13, 1999.

`Month(1011220)` returns a value of 12 because 1011220 represents December 20, 2001.

MonthFromDateTime (Reserved Word)

Disclaimer

Returns the month for the specified DateTime value.

```
MonthFromDateTime(dDateTime);
```

Where `dDateTime` is a double-precision decimal expression representing the Julian Date (since 1900) and time (since midnight).

Remarks

If a specific date is entered for the numeric expression, it must be a valid Julian date between January 1, 1901 and February 28, 2150, expressed in DateTime format.

Examples

`Value1 = MonthFromDateTime(37564.61200);` returns a value of 11 (November) for the specified DateTime of 37564.61200 (11/4/2002 2:41:17 PM).

MULTIPLE (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

MYSELF (Reserved Word)

Disclaimer

Retained for backward compatibility.

Neg (Reserved Word)

Disclaimer

Returns the absolute negative value of the number specified.

`Neg(Num)`

Where `Num` is a numeric expression representing a value to be used in the calculation.

Examples

`Neg(17)` returns -17.

`Neg(-9)` returns a value of -9.

Net_Profit_Margin (Reserved Word)

Disclaimer

Retained for backward compatibility.

NetProfit (Reserved Word)

Disclaimer

Returns the total dollar amount of all closed trades, both winning and losing.

`NetProfit`

Remarks

`NetProfit` can be computed by `GrossProfit` – `GrossLoss`.

Examples

`NetProfit` returns 1000 if there are four trades returning –500, 1200 and 300.

`NetProfit` returns 0 if there are no closed trades.

NewLine (Reserved Word)

Disclaimer

Used to start a new line when appending to a file using the FileAppend reserved word. Also used to place a carriage return / linefeed command into analysis commentary.

```
NewLine
```

Remarks

Use the + character to add `NewLine` to a string expression.

Example

```
FileAppend("c:\mydata.txt", "This symbol gapped up on " + NumToStr(Date, 0) +
NewLine);
```

NewsCount (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the number of headlines transmitted by the datafeed on the current day.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

Next (Reserved Word)

Disclaimer

Reserved word used to reference the upcoming set of price data based on the interval to which a technique is applied.

```
Buy Next Bar...
```

Remarks

`Next` is normally used with the reserved word `Bar` to specify an action to take place on the "`Next Bar`".

Examples

```
   Open of next day
```

returns the Open price of the next bar.

```
   Date of next bar
```

returns the Date of the next bar.

Additional Example

To place an order to execute one bar into the future:

```
   Buy next bar at market;
```

NoPlot (Reserved Word)

Disclaimer

This reserved word removes the specified plot from the current bar in the price chart.

```
NoPlot(Num)
```

`Num` is a numeric expression representing the number of the plot to remove.

Remarks

This reserved word is useful when collecting data on a real-time/delayed basis and you have the **Update value intra-bar** check box selected for the ShowMe/PaintBar study. If the ShowMe/PaintBar study condition becomes true during the bar, but is not true at the end of the bar, the mark is removed. If you do not use this reserved word, the mark would be placed on the bar when the condition became true and left there even when the condition was no longer true.

The value returned is a number representing the assigned width of the specified plot line.

Example

The following ShowMe study marks the low of a gap down bar, but removes the mark if the condition is no longer true for the bar:

```
If High < Low of 1 Bar Ago Then
 Plot1(Low, "GapDown")
Else
 NoPlot(1) ;
```

Additional Example

If you wanted to mark the high of a bar when the condition Close > Close[1] is true, but remove the mark if Close > Close[1] is not true for the next bar, you could use the following syntax:

```
    If Close > Close[1] Then
     Plot1(High, "CloseUp")
    Else
     NoPlot(1);
```

Not (Reserved Word)

Disclaimer

This word has been reserved for future use.

NthMaxList (Reserved Word)

Disclaimer

Returns the Nth highest value of the specified the inputs.

`NthMaxList(N, Num1, Num2, Num3)`

Where `N` is a numeric expression representing the rank number of the inputs and `Num1, Num2` and `Num3` are numeric expressions representing the values to be used in the calculation.

Remarks

A value of 3 for `N` would return the third highest value of the specified inputs.

Examples

`NthMaxList(3, 45, 72, 86, 125, 47)` returns a value of 72.

`NthMaxList(4, 18, 67, 98, 24, 65, 19)` returns a value of 24.

NthMinList (Reserved Word)

Disclaimer

Returns the Nth lowest value of the specified the inputs.

```
NthMinList(N, Num1, Num2, Num3)
```

Where `N` is a numeric expression representing the rank number of the inputs and `Num1, Num2` and `Num3` are numeric expressions representing the values to be used in the calculation.

Remarks

A value of 3 for `N` would return the third lowest value of the specified inputs.

Examples

`NthMinList(3, 45, 72, 86, 125, 47)` returns a value of 72.

`NthMinList(4, 18, 67, 98, 24, 65, 19)` returns a value of 65.

Numeric (Reserved Word)

Disclaimer

Reserved word used to define an input that expects a number passed by value.

```
InputName(Numeric);
```

Remarks

`Numeric` can be used for inputs that can be either `NumericSimple` or `NumericSeries`.

Examples

```
   Input: Price(Numeric);
```

declares the constant `Price` as a `Numeric` value to be used in a function.

```
   Input: Length(Numeric);
```

declares the constant `Length` as a `Numeric` value to be used in a function.

NumericArray (Reserved Word)

Disclaimer

Reserved word used to define an input that expects a number passed by value for each array element.

```
InputName(NumericArray);
```

Remarks

A function input is declared as a `NumericArray` when it is passing in a numeric array by value.

Examples

```
Input: PassedValues[n](NumericArray);
```

indicates that a numeric array is being passed into the function by value through the Input `PassedValues`.

NumericArrayRef (Reserved Word)

Disclaimer

Reserved word used to define an input that expects a numeric variable passed by reference for each array element.

```
InputName(NumericArrayRef);
```

Remarks

A function Input is declared as a `NumericArrayRef` when it is passing in a numeric array by reference.

Examples

```
Input: PassedValues[n](NumericArrayRef);
```

indicates that a numeric array is being passed into the function by reference through the Input `PassedValues`.

NumericRef (Reserved Word)

Disclaimer

Reserved word used to define an input that expects a numeric variable passed by reference.

```
InputName(NumericRef);
```

Remarks

A function Input is declared as a `NumericRef` when it is passing in a numeric variable by reference.

Examples

```
Input: PassedValues(NumericRef);
```

indicates that a numeric array is being passed into the function by reference through the Input `PassedValues`.

NumericSeries (Reserved Word)

Disclaimer

Reserved word used to define an input as a numeric series expression with history.

```
InputName(NumericSeries);
```

Remarks

`NumericSeries` is used for inputs whose values can be referred to historically.

Examples

```
   Input: Price(NumericSeries);
```

declares the constant `Price` as a Numeric value to be used in a function, where historical values of `Price` are available.

```
   Input: Length(NumericSeries);
```

declares the constant `Length` as a Numeric value to be used in a function, where historical values of `Length` are available.

NumericSimple (Reserved Word)

Disclaimer

Reserved word used to define an input as a numeric simple expression with no history available.

```
InputName(NumericSimple);
```

Remarks

`NumericSimple` can not be used for inputs whose values can be referred to historically.

Examples

```
   Input: Price(NumericSimple);
```

declares the constant `Price` as a Numeric value to be used in a function, restricting `Price` to be a value that does not contain historical values.

```
   Input: Length(NumericSimple);
```

declares the constant `Length` as a Numeric value to be used in a function, restricting `Length` to be a value that does not contain historical values.

NumEvenTrades (Reserved Word)

Disclaimer

Returns the number of closed-out even trades.

`NumEvenTrades`

Examples

`NumEvenTrades` returns 4 if the number of closed even trades is 4.

NUMFUTURES (Reserved Word)

Disclaimer

NUMFUTURES is an OptionStation reserved word that returns the number of future contracts in the asset section.

NUMLEGS (Reserved Word)

Disclaimer

NUMLEGS is an OptionStation reserved word that returns the number of legs in a position.

NumLosTrades (Reserved Word)

Disclaimer

Returns the number of closed-out losing trades.

`NumLosTrades`

Examples

`NumLosTrades` returns 5 if the number of closed losing trades is 5.

`NumLosTrades` returns 25 if the number of closed losing trades is 25.

NUMOPTIONS (Reserved Word)

Disclaimer

NUMOPTIONS is an OptionStation reserved word that returns the number of options in the option section.

NumToStr (Reserved Word)

Disclaimer

Converts the specified numeric expression to a string expression.

```
NumToStr(Num, Dec);
```

Where `Num` is the numeric expression that you want converted to a string expression and `Dec` is the numeric expression indicating how many decimal places to retain in the string expression.

Example

```
   NumToStr(1170.5, 2)
```

Returns the string expression "`1170.50`".

NumWinTrades (Reserved Word)

Disclaimer

Returns the number of closed-out winning trades.

```
NumWinTrades
```

Examples

`NumWinTrades` returns 5 if the number of closed winning trades is 5.

`NumWinTrades` returns 2 if the number of closed winning trades is 2

O (Reserved Word)

Disclaimer

Reserved word used to return the first price of the specified time increment or group of ticks. Abbreviation of the reserved word `Open`.

```
O
```

Remarks

Anytime the Open of a bar is needed, the letter O can be used in an equivalent fashion.

Examples

```
O of 1 bar ago
```

returns the Open price of the previous bar.

```
Average(O, 10)
```

returns the Average of the last 10 Open prices.

Additional Example

To check that the last two bars have Open prices higher than the previous bar, the following language can be used in a ShowMe study:

```
   If O > O[1] and O[1] > O[2] then
    Plot1(High, "Rising");
```

Of (Reserved Word)

Disclaimer

OF is a Skip Word that performs no function within your code. It is used within a statement only to make the EasyLanguage code more readable.

Remarks

Skip words in EasyLanguage are not necessary, however, they sometimes make it easier to understand the purpose of the code.

Syntax Coloring in the EasyLanguage PowerEditor is dark green for all skip words by default.

Example

```
If the Close of Data2 crosses over 100 Then
      Alert;
```

The word "`of`" is not necessary and the code functions the same way regardless of its presence.

On (Reserved Word)

Disclaimer

ON is a Skip Word that performs no function within your code. It is used within a statement only to make the EasyLanguage code more readable.

Remarks

Skip words in EasyLanguage are not necessary, however, they sometimes make it easier to understand the purpose of the code.

Syntax Coloring in the EasyLanguage PowerEditor is dark green for all skip words by default.

Example

```
If the Close crosses over 100 Then
       Buy next bar on Close;
```

The word "on" is not necessary and the code functions the same way regardless of its presence.

Once (Reserved Word)

Disclaimer

Reserved word used to specify that the EL statement that follows will only be executed once.

Syntax

```
Once ([optional boolean expression]) {follow by single statement or statement block}
[begin]
   Statement(s)
[end]
```

Example

```
  Once Value1 = GetAppInfo( aiApplicationType );
```

Performs the assignment of Value1 a single time. This improves performance instead of using a separate conditional expression (which must be tested on each bar) such as:

```
  if CurrentBar = 1 then Value1 = GetAppInfo( aiApplicationType );
```

Additional Examples

```
  Once (MarketPosition=1)
  Begin
   ClearPrintLog;
      Print("Log Header");
  End;
```

Clears the print log and prints a header the first time the market position equals once.

Open (Reserved Word)

Disclaimer

Reserved word used to return the first price of the specified time increment or group of ticks.

```
Open
```

Remarks

Anytime the Open of a bar is needed, the letter O can be used in an equivalent fashion.

Examples

```
Open of 1 bar ago
```

returns the Open price of the previous bar.

```
Average(Open, 10)
```

returns the Average of the last 10 Open prices.

Additional Example

To check that the last two bars have Open prices higher than the previous bar, the following language can be used in a ShowMe study:

```
  If Open > Open[1] and Open[1] > Open[2] then
   Plot1(High, "Rising");
```

OpenInt (Reserved Word)

Disclaimer

Reserved word used to return the Open Interest of the specified time increment or group of ticks.

```
OpenInt
```

Remarks

Anytime the Open Interest of a bar is needed, the letter I can be used in an equivalent fashion. See EasyLanguage Reserved Words Related to Ticks, Volume & Open Interest for more information.

Examples

```
OpenInt of 1 bar ago
```

returns the Open Interest of the previous bar.

```
Average(OpenInt, 10)
```

returns the Average of the last 10 Open Interest values.

Additional Example

To check that the last two bars have Open Interest values higher than the previous bar, the following language can be used in a ShowMe study:

```
If OpenInt > OpenInt[1] and OpenInt[1] > OpenInt[2] then
 Plot1(High, "Rising");
```

OpenPositionProfit (Reserved Word)

Disclaimer

Returns the current gain or loss of the current open position.

```
OpenPositionProfit
```

Remarks

This function can only be used in the evaluation of strategies.

Example

`OpenPositionProfit` returns a value of 86400.00 if the current open position on an S&P Futures chart had an open position profit of 86400.00.

OPTION (Reserved Word)

Disclaimer

OPTION is an OptionStation reserved word, it is a modifier to other option related reserve words.

Example

Close of Option;

OPTIONTYPE (Reserved Word)

Disclaimer

OPTIONTYPE is an OptionStation reserved word that returns the option type. (CALL = 3, Put = 2).

Or (Reserved Word)

Disclaimer

This boolean operator is used in condition statements to check multiple conditions.

```
OR
```

Remarks

`OR` requires that in order for a condition to be `True`, one or more of multiple conditions must be `True`.

Examples

```
If Plot1 Crosses Above Plot2 OR Plot2 > 5 Then...
```

`OR` is used here to determine if either the direction of the cross of the values Plot1 and Plot2, or that Plot2 is greater than 5 is `True` on the bar under consideration. If either is `True`, the condition returns `True`.

```
If Value1 Crosses Above Value2 OR Value1 > Value1[1] Then...
```

`OR` is used here to determine if either the direction of the cross of the variables `Value1` and `Value2`, or that `Value1` is greater that `Value1` of one bar ago is `True` on the bar under consideration. If either is `True`, the condition returns `True`.

Over (Reserved Word)

Disclaimer

This word is used to check for the direction of a cross between values.

```
Over
```

Remarks

Used in conjunction with the "crosses" to detect when a value crosses over, or becomes greater than another value. A value (`Value1`) crosses over another value (`Value2`) whenever `Value1` is greater than `Value2` in the current bar but Value 1 was equal to or less than Value 2 in the previous bar.

Over is a synonym for Above.

Examples

```
If Average(Close, 9) Crosses Over the Average(Close, 18) Then...
```

`Over` is used here to determine the direction of the cross of the values two moving averages on the bar under consideration.

```
If Value1 Crosses Over Value2 Then...
```

`Over` is used here to determine the direction of the cross of the variables `Value1` and `Value2` on the bar under consideration.

Pager_DefaultName (Reserved Word)

Disclaimer

Returns the string value of the default Message Recipient as specified in the Messaging tab under the File - Desktop Options menu.

```
Pager_DefaultName
```

Remarks

The string is returned in the format "Default Name".

Example

`Pager_DefaultName` returns a value of "`The Trader`" if that is the string specified in the Message Recipient box of the **Messaging** tab in the **TradeStation Desktop Preferences** dialog box.

Pager_Send (Reserved Word)

Disclaimer

Sends the specified string value to the specified string name.

```
Condition1 = Pager_Send("Str_Name", "Str_Msg") ;
```

Where `Str_Name` is a string expression which specifies the recipient of the message and `Str_Msg` is a string expression containing the message that will be sent to the pager.

`Condition1` is any true/false variable.

Remarks

Paging must be enabled and installed for delivery of the message to be completed.

Example

To obtain the true/false value returned by the reserved word, you need to assign the reserved word to a true/false variable. The following sends the message "Buy 200 AMD at Market" to Joe Trader:

```
Condition1 = Pager_Send("Joe Trader", "Buy 200 AMD at Market") ;
```

Additional Example

The `Pager_Send` reserved word can also be triggered by using the `Print` reserved word. The following also sends the message "Buy 200 AMD at Market" to Joe Trader:

```
Print( Pager_Send("Joe Trader", "Buy 200 AMD at Market") ) ;
```

PERatio (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

PercentProfit (Reserved Word)

Disclaimer

Returns the percentage of all closed-out trades that were winners.

```
PercentProfit
```

Remarks

`PercentProfit` can be calculated by `NumWinTrades` / `TotalTrades`.

Examples

`PercentProfit` returns 80 if the 8 winning trades resulted from 10 total trades.

`PercentProfit` returns 25 if the 4 winning trades resulted from 16 total trades.

Place (Reserved Word)

Disclaimer

This word is a skip word retained for backward compatibility.

Remarks

This word is skipped in EasyLanguage and is not necessary. These skipped words however, do make it easier to understand the purpose of the code.

By default, unnecessary words and sections marked as comments will appear dark green in the PowerEditor.

PlayMovieChain (Reserved Word)

Disclaimer

This reserved word plays a video clip and returns a true/false expression representing the success of the operation. If the reserved word was able to play the video clip, it returns a value of True, if it was not, it returns a value of False.

`Print(PlayMovieChain(Movie_ID));`

`Movie_ID` is a numeric expression represent ing the ID number of the video clip.

Remarks

The reference number should be a whole positive number.

Notes

Once you have created a video clip using the reserved word `MakeNewMovieRef` and added .avi files to the video clip, you are ready to play it. We recommend you use the reserved word `PlayMovieChain` only on the last bar of the chart or on bars where the commentary is obtained (using the `AtCommentaryBar` or `LastBarOnChart` reserved words). Otherwise, you may find that the video clip is played more often than you need it to.

If your intention is to play the video clip when a certain bar pattern occurs, and this pattern occurs 50 times the price chart, the trading strategy, analysis technique, or function will play the video clip 50 times when applied to the price chart.

Example

The following statements create and play a video clip on the bar where commentary is obtained:

```
Variable: ID(-1);
If BarNumber = 1 Then Begin
 ID = MakeNewMovieRef;
 Print(AddToMovieChain(ID, "c:\MyMovie.avi"));
 Print(AddToMovieChain(ID, "c:\MyOtherMovie.avi"));
End;

If LastBaronChart Then
 Print(PlayMovieChain(ID));
```

Notice that the video clip is created and the video files are added to it only once by using an `If-Then` statement to check for the first bar of the chart. If we don't use this `If-Then` statement, the indicator will create as many video clips as there are bars in the chart.

PlaySound (Reserved Word)

Disclaimer

This reserved word finds and plays the specified sound file (.wav file). This reserved word returns a value of True if it was able to find and play the sound file, and it returns a value of False if it is not able to find or play it.

```
Condition1 = (PlaySound(FileName));
```

`FileName` is any text string expression that represents the full path and file name of the sound file to be played. Only .wav files can be played.

Remarks

The specified file must be a *.wav audio file.

Notes

We recommended that you use this reserved word only on the last bar of the chart, otherwise, you may find that the .wav file is played more often than you intended. For example, if your intention is to play a .wav file whenever a certain bar pattern occurs, and this pattern occurs 50 times in the price chart, the trading strategy, analysis technique, or function will play the .wav file 50 times when it is applied to the price chart. Also, the .wav file is only played once per bar, even if the event occurs more than once intrabar (unless the **Update value intra-bar** option is enabled, in which case, the .wav file will play with each new tick while the event is True).

Example

```
   Condition1 = (PlaySound("c:\sounds\Thatsabuy.wav"));
```

This statement plays the sound file `Thatsabuy.wav` that resides in directory `c:\sounds`.

Additional Example

The following statements play the sound file `Thatsabuy.wav` when there is a key reversal pattern on the last bar of the chart:

```
   Condition1 = (PlaySound("c:\sounds\Thatsabuy.wav"));

   If LastBarOnChart AND Low < Low[1] AND Close > High[1] Then
    Condition1 = true;
```

Plot (Reserved Word)

Disclaimer

Returns the numeric value of a specified plot for use in calculations or conditional expressions.

Note This form of plot is not used to plot a value in a chart or grid. It simply returns the value of a previous plot. See Plot1-99 for the plot statement.

```
Plot(PlotName);
```

Where `PlotName` is a string expression containing the plot name.

Remarks

Once you have plotted a value using a `PlotN` statement, you can reference the value of the plot by using the reserved word `Plot` in an assignment statement or conditional expression. The reserved word `Plot` must be followed by a string expression containing the plot name enclosed in parentheses (i.e. `Plot("plot1")` or `Plot(value2)`). In addition to being a string constant, the plot name can also be an input or variable so that you can dynamically specify the plot to reference.

Example

In the indicator example below, the reserved word `Plot1` is used to plot the slow moving average and `Plot2` is used to plot the fast moving average for the `Close`. The alert condition triggers when the fast plot value crosses over the slow plot value without requiring any variables to be declared.

```
Plot1(Average(Close,18),"slow");
Plot2(Average(Close,9),"fast");
If Plot("fast") crosses over Plot("slow") then Alert("Cross Over") ;
```

Plot1-99 (Reserved Word)

Disclaimer

Used to plot values (numeric, boolean or string) in a price chart or in an grid-based analysis window. Plotted values can be standard bar values (such as Close or Volume), inputs, or variable values resulting from a calculation or an expression.

```
PlotN(Expression[,"<PlotName>"[,ForeColor[,Default[,Width]]]]);
```

N is a number between 1 and 99, representing one of the ninety-nine available plots. Expression is the value to be plotted and <PlotName> is the name of the plot. ForeColor is an Easy Language color used for the plot (also referred to as PlotColor), Default (reserved for future use), and Width is a numeric value representing the width of the plot. The parameters <PlotName>, ForeColor, Default, and Width are optional.

Notes

The parameter Default currently has no effect. However, a value for the parameter is required in order to specify a width, as discussed in the example.

Remarks

The reserved word Plot can also be used in an assignment or condition to refer to a plotted value. When used this way, the plot name must appear in parentheses immediately following the reserved word (i.e. Plot("PlotName"). Refer to Plot (Reserved Word) for more information..

Example 1

Any one or more of the optional parameters can be omitted, as long as there are no other parameters to the right. For example, the Default and Width parameters can be excluded from a statement as follow:

```
Plot1(Volume, "V", Red);
```

But the plot name cannot be omitted if you want to specify the plot color and width. For instance, the following example generates a syntax error because the name of the plot statement is expected:

Incorrect:

```
Plot1(Volume, Black, Default, 2);
```

Correct:

```
Plot1(Volume, "V", Black, Default, 2);
```

The only required parameter for a valid Plot statement is the value to be plotted. So the following statement is valid:

```
Plot1(Volume);
```

When no plot name is specified, EasyLanguage uses Plot1, Plot2.. Plot99 as the plot names for each plot. The first plot is named Plot1, the second Plot2, and so on.

Whenever referring to the plot color or width, you can use the word Default in place of the parameter(s) to have the Plot statement use the default color and/or width selected in the **Properties** tab of the **Format indicator** dialog box.

For example, the following statement will display the volume in the default color but indicates a specific width:

```
Plot1(Volume, "V", Default, Default, 3);
```

Again, you can use the word Default for the color parameters or the width parameter.

Also, the same plot (that is, Plot1, Plot2) can be used more than once in an analysis technique; the only requirement is that you use the same plot name in both instances of the Plot statement. If no name is assigned, then the default plot name is used (that is, Plot1, Plot2).

For example, if you want to plot the net change using red when it is negative and green when it is positive, you can use the same plot number (in this case Plot1) twice, as long a the name of the plot is the same:

```
Value1 = Close - Close[1];

If Value1 > 0 Then
 Plot1(Value1, "NetChg", Green)
Else
 Plot1(Value1, "NetChg", Red);
```

In this example, the plot name "NetChg" must be the same in both instances of the Plot statement.

Example 2

When applying the analysis technique to a chart, you can displace the plot to the right or left. For example:

```
Plot1[3](Value1);
```

The above example calculates the plot value using the current bar but draws it on the chart 3 bars ago. Use a negative number to draw the value 3 bars ahead of the current bar.

Example3

You can plot a text string (enclosed in double quotes) or a boolean value (true/false) to a cell in a grid-based window or on the status line of a sub-graph in a chart. The second parameter label is optional and becomes the column heading when used with a gird-based application. For example:

```
Plot1("Up","Upward Value", Green);
```

In the above example the word 'Up' is displayed in green within a grid cell or on the sub-graph status line.

Example4 - Getting Plot Value (Refer to Plot (Reserved Word) for more information)

Once you have defined a plot using the `PlotN` statement, you can reference the value of the plot simply by using the reserved word `Plot` in a calculation or condition. In the example below, the `Plot1` statement is used to plot the accumulation distribution of the volume. The value of the plot is referenced in the next statement, in order to write the alert criteria:

```
Plot1(AccumDist(Volume), "AccumDist") ;
If Plot("AccumDist") > Highest(Plot("AccumDist"), 20) then Alert ;
```

PlotPaintBar (Reserved Word)

Disclaimer

This reserved word is used only within a PaintBar study, and enables you to paint the entire bar a specified color or paint the bar between two specified values.

```
PlotPaintBar( BarHigh, BarLow [, BarOpen [, BarClose [, "<PlotName>"[, ForeColor[,
Default[, Width]]]]]] );
```

`BarHigh`, `BarLow`, `BarOpen` and `BarClose` are numeric expressions representing the high, low, open and closing prices for the bar to be drawn by the PaintBar study, and <`PlotName`> is the name of the plot. `ForeColor` is an Easy Language color used for the plot (also referred to as `PlotColor`), `Default` (reserved for future use), and `Width` is a numeric value representing the width of the plot.

Remarks

You can also specify only two of the bar parameters instead of the four: `BarHigh`, `BarLow`, `BarOpen` or `BarClose`. However, you must specify either two or four of the bar parameters.

EasyLanguage will automatically append a number from 1 – 4 to each of the four plot statements represented by the statement, so as to be able to individually identify each plot name in EasyLanguage. If the maximum number of characters for the `PlotName` is used, the last character in the string will be dropped and replaced with the appropriate number for each plot. If the `PlotName` parameter is not included, the plots will be assigned the generic plot names Plot1 – Plot4.

You can abbreviate the `PlotPaintBar` reserved word to `PlotPB`. Also, you can use the `Plot` reserved word described previously to write a PaintBar study; however, we rec ommend you use the `PlotPaintBar` reserved word.

When applying the analysis technique to a chart, you can displace the plot to the right or left. For example:

```
   PlotPaintBar[3](Value1);
```

The above example calculates the plot value using the current bar but draws it on the chart 3 bars ago. Use a negative number to draw the value 3 bars ahead of the current bar.

Notes

The parameter `Default` currently has no effect. However, a value for the parameter is required in order to specify a width, as discussed in the example.

Example

For example, the following instructions can be used in order to paint red the bars with twice the 10-bar average of the volume:

```
If Volume > 2 * Average(Volume, 10) Then
 PlotPB(High, Low, Open, Close, "AvgVol", Red );
```

Additional Example

The following instructions paint the area between the two plots of the Bollinger Bands Indicator when the 14-bar ADX value is lower than 25:

```
Variables: Top(0), Bottom(0);

Top = BollingerBand(Close, 14, 2);
Bottom = BollingerBand(Close, 14, -2);

If ADX(14) < 25 Then
 PlotPaintBar(Top, Bottom, "Area", Blue);
```

In this last example, notice that although we omitted the `BarOpen` and `BarClose` parameters, we are still able to specify the name and color of the plot.

PlotPB (Reserved Word)

Disclaimer

This reserved word is an abbreviated version of PlotPaintBar and enables you to paint the entire bar a specified color or paint the bar between two specified values.

```
PlotPB( BarHigh, BarLow [, BarOpen [, BarClose [, "<PlotName>"[, ForeColor[, Default[,
Width]]]]]] );
```

BarHigh, BarLow, BarOpen and BarClose are numeric expressions representing the high, low, open and closing prices for the bar to be drawn by the PaintBar study, and <PlotName> is the name of the plot. ForeColor is an Easy Language color used for the plot (also referred to as PlotColor), Default (reserved for future use), and Width is a numeric value representing the width of the plot.

Remarks

You can also specify only two of the bar parameters instead of the four: BarHigh, BarLow, BarOpen or BarClose. However, you must specify either two or four of the bar parameters.

EasyLanguage will automatically append a number from 1 – 4 to each of the four plot statements represented by the statement, so as to be able to individually identify each plot name in EasyLanguage. If the maximum number of characters for the PlotName is used, the last character in the string will be dropped and replaced with the appropriate number for each plot. If the PlotName parameter is not included, the plots will be assigned the generic plot names Plot1 – Plot4.

You can use the Plot reserved word described previously to write a PaintBar study; however, we rec ommend you use the PlotPaintBar reserved word.

When applying the analysis technique to a chart, you can displace the plot to the right or left. For example:

```
   PlotPB[3](Value1);
```

The above example calculates the plot value using the current bar but draws it on the chart 3 bars ago. Use a negative number to draw the value 3 bars ahead of the current bar.

Notes

The parameter Default currently has no effect. However, a value for the parameter is required in order to specify a width, as discussed in the example.

Example

For example, the following instructions can be used in order to paint red the bars with twice the 10-bar average of the volume:

```
If Volume > 2 * Average(Volume, 10) Then
 PlotPB(High, Low, Open, Close, "AvgVol", Red );
```

Additional Example

The following instructions paint the area between the two plots of the Bollinger Bands Indicator when the 14-bar ADX value is lower than 25:

```
Variables: Top(0), Bottom(0);

Top = BollingerBand(Close, 14, 2);
Bottom = BollingerBand(Close, 14, -2);

If ADX(14) < 25 Then
 PlotPaintBar(Top, Bottom, "Area", Blue);
```

In this last example, notice that although we omitted the BarOpen and BarClose parameters, we are still able to specify the name and color of the plot.

PM_GetCellValue (Reserved Word)

Disclaimer

This reserved word returns the number corresponding to the value of the specified cell of the ProbabilityMap study drawing area. The number returned by this reserved word is between 0 and 100**.**

```
PM_GetCellValue(Column, Price)
```

`Column` and `Price` are numeric expressions representing the cell in the ProbabilityMap study drawing area for which you want to obtain the value. To obtain the value returned by this reserved word, you can assign the value to a numeric variable, for example, `Value1`.

Remarks

- The return value represents the intensity of the cell. Higher values have a greater intensity and lower values have a lesser intensity. Intensity is reflected by color.
- The column value must be greater than zero, but cannot exceed the number of columns in the ProbabilityMap.
- The price value must be a valid price within the ProbabilityMap grid.

Example

The following statement obtains the value of the cell in the lower left corner of the Proba bilityMap study drawing area:

```
Value1 = PM_GetCellValue (1, PM_Low);
```

PM_GetNumColumns (Reserved Word)

Disclaimer

This reserved word returns a numeric value representing the number of columns of the ProbabilityMap study drawing area.

```
Value1 = PM_GetNumColumns
```

Notes

To obtain the value returned by this reserved word, you can assign the value to a numeric variable, for example, `Value1`.

Example

If you wanted to set the variable, `Value1`, equal to the number of columns in the ProbabilityMap array, you could use the following syntax:

```
   Value1 = PM_GetNumColumns;
```

Additional Example

The following loop traverses a row of the ProbabilityMap study drawing area from the first to the last column:

```
For Value1 = 1 To PM_GetNumColumns Begin
 { EasyLanguage instruction(s) }
End;
```

PM_GetRowHeight (Reserved Word)

Disclaimer

This reserved word returns numeric value representing the height (in points) of the cells of the ProbabilityMap study drawing area.

```
PM_GetRowHeight
```

Notes

To obtain the value returned by this reserved word, you can assign the value to a numeric variable, for example, `Value1`.

Example

`PM_GetRowHeight` returns the height of the rows in the ProbabilityMap.

Additional Example

The following loop traverses the first column of the ProbabilityMap study drawing area:

```
Value1 = PM_Low;
While Value1 < PM_High Begin
 { EasyLanguage instructions }
 Value1 = Value1 + PM_GetRowHeight;
End;
```

PM_High (Reserved Word)

Disclaimer

This reserved word returns a numeric value representing the upper boundary of the ProbabilityMap study drawing area. This value is important to ensure that you don't query the values outside of the ProbabilityMap study drawing area.

```
PM_High
```

Remarks

The return is based on the upper extreme of the ProbabilityMap grid, not necessarily the extreme of the colored cells.

Example

```
PM_High
```

Returns the upper extreme of the ProbabilityMap study.

Example

The following statement checks whether or not a particular value is inside the upper and lower boundaries of the ProbabilityMap study drawing area before assigning a color value to a cell:

```
If Value1 >= PM_Low AND Value1 <= PM_High Then
 PM_SetCellValue(1, Value1, 100);
```

PM_Low (Reserved Word)

Disclaimer

This reserved word returns a numeric value representing the lower boundary of the ProbabilityMap study drawing area. This value is important to ensure that you don't query values outside of the ProbabilityMap study drawing area.

```
PM_Low
```

Remarks

The return value is based on the lower extreme of the ProbabilityMap grid, not necessarily the extreme of the colored cells.

Example

```
PM_Low
```

Returns the lower extreme of the ProbabilityMap study.

Additional Example

The following statement checks whether or not a particular value is inside the upper and lower boundaries of the ProbabilityMap study drawing area before assigning a color value to a cell:

```
If Value1 >= PM_Low AND Value1 <= PM_High Then
 PM_SetCellValue(1, Value1, 100);
```

PM_SetCellValue (Reserved Word)

Disclaimer

This reserved word is used to set the location and intensity of individual cells in the ProbabilityMap drawing area.

```
PM_SetCellValue(Column, Price, Value)
```

`Column, Price`, and `Value` are numeric expressions. `Column` and `Price` are the column and the row of the drawing area, respectively, and `Value` is a numeric expression between 0 and 100 that colors that particular cell.

Remarks

- The column value must be within the allocated range — greater than 0 and less than the NumColumns setting.
- The price value must be within the allocated range — greater than the low range extreme and less than the upper range extreme.
- Meaningful value settings exist only between 0 and 100.

Example

The following statement sets the cell in the column corresponding to the close of the last bar on the chart (the first bar in the ProbabilityMap drawing area) to a value of 100:

```
PM_SetCellValue(1, Close, 100);
```

Additional Examples

```
PM_SetCellValue(1, 100, 90);
```

Adds a ProbabilityMap cell to the first column of the grid at the price level of 100, with an intensity setting of 90, thus producing a whitish colored cell.

```
PM_SetCellValue(5, 80, 10);
```

Adds a ProbabilityMap cell to the fifth column of the grid at the price level of 80, with an intensity setting of 10, thus producing a dark red colored cell.

PM_SetHigh (Reserved Word)

Disclaimer

This reserved word specifies the upper boundary of the ProbabilityMap area. The ProbabilityMap is not drawn above the value specified.

```
PM_SetHigh(Num)
```

`Num` is a numeric expression representing the upper boundary of the ProbabilityMap study.

Example

The following statement sets the upper boundary of the ProbabilityMap study to a value of the close plus three times the range of the current bar:

```
   PM_SetHigh(Close + (Range * 3));
```

`PM_SetHigh` sets the upper range value of a ProbabilityMap.

Additional Example

```
   PM_SetHigh(Close + (2 * StdDev(Close, 10)));
```

sets a ProbabilityMap's upper value to the close, plus two standard deviations.

PM_SetLow (Reserved Word)

Disclaimer

This reserved word specifies the lower boundary of the ProbabilityMap area. The ProbabilityMap is not drawn below the value specified.

`PM_SetLow(Num)`

`Num` is a numeric expression representing the lower boundary of the ProbabilityMap.

Example

The following statement sets the lower boundary of the ProbabilityMap study to the a value equal to the lowest low of the last 20 bars:

```
PM_SetLow( Lowest(Low, 20) );
```

Additional Example

```
PM_SetLow(Close - (2 * StdDev(Close, 10)));
```

Sets a ProbabilityMap's lower value to the close, minus two standard deviations.

PM_SetNumColumns (Reserved Word)

Disclaimer

This reserved word is used to determine the number of columns inside the ProbabilityMap drawing area. The ProbabilityMap is not drawn past the number of bars specified.

```
PM_SetNumColumns(Num)
```

`Num` is a numeric expression representing the maximum number of bars to the right of the current bar that the ProbabilityMap is to be drawn.

Example

The following statement defines the ProbabilityMap study drawing area to 50 bars:

```
   PM_SetNumColumns(50);
```

Additional Example

You can use the following expression to set the ProbabilityMap drawing area to have as many columns as bars to the right available in the chart:

```
   PM_SetNumColumns(MaxBarsForward);
```

PM_SetRowHeight (Reserved Word)

Disclaimer

This reserved word is used to specify (in points) the height of each row of the ProbabilityMap drawing area.

```
PM_SetRowHeight(Num)
```

`Num` is a numeric expression representing the row height.

Notes

The row height of the drawing area is usually specified as:

(ProbabilityMap Upper Boundary - ProbabilityMap Lower Boundary) / Number of Rows

So, for instance, if the difference between the upper and lower boundaries of the Probabil ityMap is 50, and you want 100 rows, the row height must be 0.5. The more rows there are in the ProbabilityMap, the better 'resolution'; in other words, the grid cells are smaller and the resulting graph appears smother and more detailed. However, it takes more time to draw, as there are more cells to calculate and for which to draw ProbabilityMap values.

Example

If you want to have 50 rows in the ProbabilityMap, the following instructions specify the appropriate row height:

```
PM_SetRowHeight((PM_High - PM_Low) / 50 );
```

Additional Examples

```
PM_SetRowHeight(.125);
```

Sets the ProbabilityMap cell increment to .125.

```
PM_SetHigh(MinMove * (1 / PointValue));
```

Sets the ProbabilityMap cell increment equivalent to the minimum movement of the instrument.

Pob (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word Limit.

Point (Reserved Word)

Disclaimer

Returns a decimal numeric expression equivalent to 1 divided by the PriceScale (1 / PriceScale).

```
Point
```

Remarks

A `Point` represents the smallest fractional unit of PriceScale. For example, for stocks with a PriceScale of 100, a point would be .01 (or 1/100). For instruments that trade in a PriceScale of 32nds, a point would be 0.03125 (or 1/32).

Note A point is not the same as the minimum price move (which is MinMove / PriceScale). For example, for the E-Mini with a PriceScale of 100, a point is .01 (or 1/100) while its minimum price move is .25 (MinMove / PriceScale)

Examples

```
   Close + 1 Point
```

returns one increment of the PriceScale added to the Close price.

```
   Low - 1 Point
```

returns the Low of the bar minus one increment of the PriceScale.

Additional Example

An exit statement can be written to prevent large losses by:

```
   Sell This Bar at EntryPrice - 1 Point Stop;
```

POINTER (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

PointValue (Reserved Word)

Disclaimer

Returns a numeric expression representing the market value of one
PriceScale unit (defined as a Point is EasyLanguage) for a particular symbol.
PointValue is equivalent to BigPointValue / PriceScale. A stock would have a
PointValue of .01.

> **Note** The PointValue is not always the same as the minimum move value. To calculate the minimum move value for a share or contract, multiply PointValue by MinMove. For example, for the E-Mini S&P with a PointValue of .50, the minimum move value is 12.50 (MinMove * PointValue) based on a minimum price move of .25 per contract .

```
PointValue
```

Examples

`PointValue * MinMove` returns the market value of the minimum price move.

`PointValue * PriceScale` returns the market value for a full point price move.

See Also

PriceScale

MinMove

BigPointValue

Pos (Reserved Word)

Disclaimer

Returns the absolute value of num.

`Pos(Num)`

Where `Num` is a numeric expression to be used in the calculation.

Examples

`Pos(-5)` returns 5.

`Pos(350)` returns 350.

POSITION (Reserved Word)

Disclaimer

POSITION is an OptionStation reserved word, it is a modifier to other position related reserve words.

Example

MaxGain of Position;

POSITIONID (Reserved Word)

Disclaimer

POSITIONID is an OptionStation reserved word, that returns a value for an identifiable position.

POSITIONID return values:
Long Calls = 1
Long Puts = 2
Write Calls = 3
Write Puts = 4
Bull/Call Debit Spread = 5
Bear/Put Debit Spread = 6
Bull/Put Credit Spread = 7
Bear/Call Credit Spread = 8
Long Straddle = 9
Long Strangle = 10
Short Straddle = 11
Short Strangle = 12
Long Call Butterfly = 13
Long Put Butterfly = 14
Call Time Spread = 15
Put Time Spread = 16
Coverered Call = 17
Married Put = 18
Call Ratio Spread = 19
Put Ratio Spread = 20
Call Ratio Backspread = 21
Put Ratio Backspread = 22
Long Stock = 23
Long Future = 24
Short Stock = 25
Short Future = 26
Synthetic Long = 27
Synthetic Short = 28
Call Iron Butterfly = 29
Put Iron Butterfly = 30
Long Iron Butterfly = 31
Short Iron Butterfly = 32
Short Call Condor = 33
Short Put Condor = 34
Long Call Condor = 35
Long Put Condor = 36
Short Call Butterfly = 37
Short Put Butterfly = 38
Put Hedge = 39
Synthetic Put = 40
Or Cannot Identify

PositionProfit (Reserved Word)

Disclaimer

Returns the current gain or loss of the specified position..

`PositionProfit(Num);`

`Num` is a numeric expression representing the number of positions ago (up to a maximum of ten).

Remarks

This function can only be used in the evaluation of strategies. It does not require an input, however, by using the input `Num`, you can obtain the specified value from a previous position, up to ten positions ago.

Example

`PositionProfit(3)` returns a value of -1.00 if the third most recent position on a chart had a loss of 1.00.

POSITIONSTATUS (Reserved Word)

Disclaimer

Retained for backward compatibility.

Power (Reserved Word)

Disclaimer

Returns the number raised to the specified power.

```
Power(Num, Pow);
```

Where `Num` and `Pow` are the number to raise to the specified power and the power by which to raise the number, respectively.

Examples

`Pow(2,3)` returns 8

`Pow(4,5)` returns 1024.

PrevClose (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the close of the previous trading session.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

PrevOpenInt (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the open Interest of the previous trading day.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

PrevVolume (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

Price_To_Book (Reserved Word)

Disclaimer

Retained for backward compatibility.

PriceScale (Reserved Word)

Disclaimer

Returns a numeric expression representing the fractional portion of a full point move for a particular symbol. The PriceScale value is represented by a whole number. For example; the PriceScale of the E-Mini S&P is100, which represents the fractional portion of a full point move for the E-Mini S&P. Any stock would also have a PriceScale of 100. 1 / PriceScale = Minimum price increment.

```
PriceScale
```

Examples

`MinMove / PriceScale` returns the smallest increment between trades.

`MinMove / PriceScale * BigPointValue` returns the dollar value of the smallest change in price.

See Also

PointValue

MinMove

BigPointValue

Print (Reserved Word)

Disclaimer

Sends information to the Print Log in the EasyLanguage PowerEditor or, if specified, to an output location (a file or the default printer).

```
Print(Parameters);
```

`Parameters` is any number of valid string, true/false, numeric series, or numeric expressions, each separated by a comma. To send output to a printer or file instead of the Print Log, you must also specify output location. The maximum number of displayed characters supported for the combined `Parameters` is 255.

Remarks

If no output location is specified, the information is sent to the Print Log in the EasyLanguage PowerEditor.

To send information to a specified file instead of the Print Log, specify the file and path as the first parameter. Enclose the file name and path in quotation marks. To send the information to the default printer, include the word Printer as the first parameter.

You can format the numeric expressions displayed using the `Print` reserved word. To do so, use the following syntax:

```
   Print(Value1:N:M);
```

`Value1` is any numeric expression, `N` is the minimum number of integers to use, and `M` is the number of decimals to use. If the numeric expression being sent to the Print Log has more integers than what is specified by `N`, the `Print` statement uses as many digits as necessary, and the decimal values are rounded to the nearest value. For example, assume `Value1` is equal to 3.14159 and we have written the following statement:

```
   Print(Value1:0:4);
```

The numeric expression displayed in the Print Log would be 3.1416. As another ex ample, to format the closing prices, you can use the following statement:

```
   Print(ELDateToString(Date), Time, Close:0:4);
```

Examples

The following statement sends the date, time, and closing price of the current bar to the Print Log:

```
   Print(Date, Time, Close);
```

The following statement sends the same information to an ASCII file called Mydata.txt:

```
   Print(File("c:\data\mydata.txt"), Date, Time, Close);
```

The following statement sends the information to the default printer instead:

```
   Print(Printer, Date, Time, Close);
```

Printer (Reserved Word)

Disclaimer

Sends information to the default printer from a Print statement.

```
Print(Printer, Parameters);
```

`Parameters` is any number of valid string, numeric series, or numeric expression, each separated by a comma. To send output to a printer or file instead of the Print Log, you must also specify output location.

Remarks

The word "`Printer`" must be the first expression listed in the print statement followed by a comma, and then the requested information. Commas must be used to separate multiple expressions in a print statement.

Examples

If you wanted to send the date, time, and close information to the printer when you applied an analysis technique, you could use the following syntax:

```
   Print(Printer, Date, Time, Close);
```

Product (Reserved Word)

Disclaimer

Returns the number representing the product being used.

```
Product
```

Remarks

The Product function returns values based on the following table.

Product Name	Product Number
TradeStation	0

Examples

If you wanted to only display an indicator when the user applied the indicator to a TradeStation chart, you could use the following syntax:

```
If Product = 0 Then
 Plot1(Value1, "Indicator");
```

PROFIT (Reserved Word)

Disclaimer

Retained for backward compatibility.

ProfitCF (Reserved Word)

Disclaimer

Reserved word that returns a profit currency correction factor that may be used for calculating a profit when a trade is exited.

Remarks

`ProfitCF` is used to apply a currency conversion factor to profit calculations when a symbol reports its value in a currency that is different from that of the clients trading account.

See About Currency Conversion for information about currency conversion factors.

ProfitTargetStop (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word SetProfitTarget.

PROTECTIVE (Reserved Word)

Disclaimer

Retained for backward compatibility.

PUT (Reserved Word)

Disclaimer

PUT is a constant reserve word returning the value 2.

PUTCOUNT (Reserved Word)

Disclaimer

PUTCOUNT is an OptionStation reserved word, referring to the number of puts in the OptionStation option(n) array.

PUTITMCOUNT (Reserved Word)

Disclaimer

PUTITMCOUNT is an OptionStation reserved word, reserved for future use.

PutOpenInt (Reserved Word)

Disclaimer

Returns the daily total Put option open interest of all the options for an underlying stock, index, or future. Allows historical bars back reference.

Example

```
Value1 = PutOpenInt;
```

PUTOTMCOUNT (Reserved Word)

Disclaimer

PUTOTMCOUNT is an OptionStation reserved word, reserved for future use.

PUTSERIESCOUNT (Reserved Word)

Disclaimer

PUTSERIESCOUNT is an OptionStation reserved word, reserved for future use.

PUTSTRIKECOUNT (Reserved Word)

Disclaimer

PUTSTRIKECOUNT is an OptionStation reserved word, reserved for future use.

PutVolume (Reserved Word)

Disclaimer

Returns the daily total Put option volume of all the options for an underlying stock, index, or future. Allows historical bars back reference.

Example

```
Value1 = PutVolume;
```

q_Ask (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the current best ask for a symbol.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

q_AskExchange (Reserved Word)

Disclaimer

A quote field that returns a text string representing the exchange the last ask was sent from.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

q_AskSize (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by AskSize.

q_Bid (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by InsideBid.

q_BidExchange (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by BidExch.

q_BidSize (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by BidSize.

q_BigPointValue (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by BigPointValue.

q_CallOpenInt (Reserved Word)

Disclaimer

A quote field that returns the daily total Call option open interest of all the options for an underlying stock, index, or future from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

q_CallVolume (Reserved Word)

Disclaimer

A quote field that returns the daily total Call option volume of all the options for an underlying stock, index, or future from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

q_Category (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by Category.

q_Close (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

q_DailyLimit (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by DailyLimit.

q_Date (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by TradeDate.

q_DateEX (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by TradeDateEX.

q_DateLastAsk (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by AskDate.

q_DateLastAskEX (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by AskDateEX.

q_DateLastBid (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by BidDate.

q_DateLastBidEX (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by BidDateEX.

q_DateLastTrade (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by TradeDate.

q_DateLastTradeEX (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by TradeDateEX.

q_DownVolume (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by DailyVolumeDown.

q_ExchangeListed (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by ExchListed.

q_ExpirationDate (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by ExpDate.

q_ExpirationDateEX (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by ExpDateEX.

q_High (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

q_IVolatlity (Reserved Word)

Disclaimer

A quote field that returns the daily average weighted Implied Volatility of the options for an underlying stock, index, or future.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

q_Last (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by Last.

q_LastTradingDate (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by LTD.

q_LastTradingDateEX (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by LTDEX.

q_Low (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

q_Margin (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by Margin.

q_MinMove (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by MinMove.

q_MinutesDelayed (Reserved Word)

Disclaimer

Note This reserved word is retained for backward compatibility only.

Returns a numeric expression representing the number of minutes this symbol is delayed.

q_NewsCount (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by NewsCount.

q_NumOptions (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by NumOptions.

q_Offer (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by InsideAsk.

q_Open (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

q_OpenInterest (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by CurrentOpenInt.

q_OptionType (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by OptionType.

q_PreviousClose (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by PrevClose.

q_PreviousDate (Reserved Word)

Disclaimer

Note This reserved word is retained for backward compatibility only.

Returns a numeric expression representing the EasyLanguage date of the previous trading session.

q_PreviousOpenInterest (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by PrevOpenInt.

q_PreviousTime (Reserved Word)

Disclaimer

Note This reserved word is retained for backward compatibility only.

Returns a numeric expression representing the time of the previous trading day.

q_PreviousVolume (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by PrevVolume.

q_PrevTotalVolume (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

q_PutOpenInt (Reserved Word)

Disclaimer

A quote field that returns the daily total Put option open interest of all the options for an underlying stock, index, or future from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

q_PutVolume (Reserved Word)

Disclaimer

A quote field that returns the daily total Put option volume of all the options for an underlying stock, index, or future from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

q_StrikePrice (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the strike price of an option..

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

q_Time (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by TradeTime.

q_TimeEX (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by TradeTimeEX.

q_TimeLastAsk (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by AskTime.

q_TimeLastAskEX (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by AskTimeEX.

q_TimeLastBid (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by BidTime.

q_TimeLastBidEX (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by BidTimeEX.

q_TimeLastTrade (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by TradeTime.

q_TimeLastTradeEX (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by TradeTimeEX.

q_TotalVolume (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by DailyVolume.

q_TradeVolume (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by TradeVolume.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

q_UnchangedVolume (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by DailyVolumeUC.

q_UpVolume (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by DailyVolumeUp.

q_Yield (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by DivYield.

Quick_Ratio (Reserved Word)

Disclaimer

Retained for backward compatibility.

RaiseRunTimeError (Reserved Word)

Disclaimer

Causes a user specified run-time error message to be generated.

Syntax

`RaiseRunTimeError(strReason)`

`strReason` is a string expression that contains the text of the user specified error message..

Examples

`RaiseRunTimeError("My Own Error Message")` will generate an error and will display the specified message in the events log.

Random (Reserved Word)

Disclaimer

Returns a random number in the range between 0 and `Num`.

`Random(Num)`

`Num` is a numeric expression to be used in the calculation.

Examples

`Random(35)` might return a value of 21.26.

`Random(142.56)` might return a value of 136.23.

RAWASK (Reserved Word)

Disclaimer

RAWASK is an OptionStation reserved word, referring to the modeled current best ask value of an option or underlying asset.

Example

RawAsk of option;

RawAsk of asset;

RAWBID (Reserved Word)

Disclaimer

RAWBID is an OptionStation reserved word, referring to the modeled current best bid value of an option or underlying asset.

Example

RawBid of option;

RawBid of asset;

Red (Reserved Word)

Disclaimer

Sets the plot color or background color to Red.

```
Red
```

Remarks

There are 16 colors available in EasyLanguage. For more information, see EasyLanguage Colors and Their Corresponding Numeric Values.

Examples

```
  Plot1(Value1, "Test", Red)
```

plots `Value1` with the name `Test`, and sets the color of `Plot1` to `Red`.

```
  TL_SetColor(1, Red)
```

sets the color of a TrendLine with a reference number of 1 to `Red`.

REGULARSESSION (Reserved Word)

Disclaimer

REGULARSESSION is used with session information to refer to regular sessions.

Repeat/Until (Reserved Word)

Disclaimer

Evaluates one or more statements in a loop and exits the loop only when the until condition is true.

Syntax

```
Repeat
  statement(s);
Until (condition_is_true);
```

Remarks

There is no need to use the begin/end keywords to group more than one program statement, as all statements between repeat and until are treated as a block.

The repeat statement is similar to the while loop, however, with the repeat statement, the conditional test occurs after the loop. This means that the program statement(s) which constitute the loop body will be executed at least once.

Ret_On_Avg_Equity (Reserved Word)

Disclaimer

Retained for backward compatibility.

RevSize (Reserved Word)

Disclaimer

Returns the Reversal size of a Point & Figure chart.

```
RevSize
```

Remarks

The Reversal size of a Point & Figure chart is defined by the user on the Format Symbol dialog, Settings tab.

Examples

`RevSize` returns 3 if the reversal size is set to 3.

`RevSize` returns 1 if the reversal size is set to 1.

RGB (Reserved Word)

Disclaimer

Returns an RGB color (16 million) value as a 32-bit integer that represents a specified combination of red, green, and blue color values.

```
RGB(RedValue, GreenValue, BlueValue);
```

Where each red, blue, and green color value can be an integer from 0-255.

Remarks

The value returned is a RGB color number (16 million color) from 0 to 16777215.

Example

If you wanted to change the foreground color of Plot1 to yellow, you could use the following syntax:

```
  SetPlotColor (1, RGB (255,255,0));
```

RGBToLegacyColor (Reserved Word)

Disclaimer

Returns an EasyLangauge legacy color value from 1 to 16 that most closely matches the specified RGB (16 million) color value.

```
RGBToLegacyColor(RGBColorValue);
```

Where RGBColorValue is an EasyLangauge RGB color number from 0 to 16777215.

Remarks

The value returned is an integer value ranging from 1-16.

Example

To following returns a legacy color value of 2 (blue) for an RGB value of 16711682.

```
   Value1 = RGBToLegacyColor(16711682);
```

RHO (Reserved Word)

Disclaimer

RHO is used in OptionStation to return the risk value rho of an option or position.

RightSide (Reserved Word)

Disclaimer

This reserved word is used with the other ActivityBar reserved words to specify the side of the ActivityBar you want to reference. It specifies that you are referencing the right side of the bar.

```
RightSide
```

Example

The following statement obtains the number of cells on the right side of a bar, for the row corresponding to the open price:

```
Value1 = AB_GetNumCells(Open of Data1, RightSide);
```

RightStr (Reserved Word)

Disclaimer

Reduces the specified string expression.

```
RightStr(Str1, sSize) ;
```

`Str1` is the string expression that you want to reduce; it must be enclosed in quotation marks. `sSize` is the numeric expression indicating the number of characters, counting from the end of the string expression, that will be retained. Any additional characters are removed.

Example

`RightStr("Net Profit", 6)` returns the string expression "Profit".

Round (Reserved Word)

Disclaimer

Returns `Num` rounded to `Prec` decimal places.

```
Round(Num, Prec)
```

`Num` is a numeric expression to be used in the calculation and `Prec` is a numeric expression representing the number of decimal places to keep.

Examples

`Round(142.3215, 2)` returns a value of 142.32.

`Round(9.5687, 3)` returns a value of of 9.569.

RunCommand (Reserved Word)

Disclaimer

See RunCommandOnLastBar for more information.

Saturday (Reserved Word)

Disclaimer

Returns the number 6 as the value for Saturday.

```
Saturday
```

Example

```
   Saturday
```

…returns a value of 6 as the value for Saturday.

Screen (Reserved Word)

Disclaimer

This word has been reserved for future use.

SDECOMMENTARY (Reserved Word)

Disclaimer

Reserved for future use.

SecondsFromDateTime (Reserved Word)

Disclaimer

Returns the numeric value of seconds for the specified DateTime.

```
SecondsFromDateTime(dDateTime);
```

Where `dDateTime` is a double-precision decimal expression representing the Julian Date (since 1900) and time (since midnight).

Remarks

If a specific date is entered for the numeric expression, it must be a valid Julian date between January 1, 1900 and February 28, 2150, expressed in DateTime format.

Examples

`Value1 = SecondsFromDateTime(37564.61200);` returns a minutes value of 17 for the specified DateTime of 37564.61200 (11/4/2002 2:41:17 PM).

Sell (Reserved Word)

Disclaimer

The Sell order is used to close a long position. The order specifications are defined by parameters in the Sell statement.

Exit orders do not pyramid. Once the exit criteria are met and the exit order filled, the order is ignored for that position until the position is modified (that is, more shares/ contracts are bought or a new long position is established).

```
Sell [("Order Name")] [from entry ("Entry Name")] [ Number of Shares [Total]]
    [Execution Method];
```

Only the word `Sell` and the `Order Action` are required by the order syntax rules. For example:

```
Sell this bar on close;
```

or

```
Buy next bar at 45 stop;
```

If no other parameters are specified, the default value for the `Order Name` is `"Sell"`, and all long contracts will be exited from the position.

Each portion of the statement, `Order Name`, `Number of Shares`, `from entry` and `Execution Method` are described next.

Order Name

When using multiple exits within a strategy, it is helpful to label each exit order with a different name. This enables you to identify these exit orders in both the price chart and the Strategy Performance Report. To assign an exit order a name, specify a name in quotation marks and within parentheses immediately after the word `Sell`. For Example:

```
Sell ("My Exit") This Bar on Close;
```

This instruction closes the entire long position, and the order is labeled `My Exit`.

Tying an Exit to an Entry

It is possible to tie an exit instruction to a specific entry. This can be achieved only if you named the long entry, and the long entry is in the same strategy as the exit order. Consider the following strategy:

```
Buy ("MyBuy") 10 Shares Next Bar at Market;
Buy 20 Shares Next Bar at High + 1 Point Stop ;

Sell From Entry ("MyBuy") Next Bar at High + 3 Points Stop;
```

In the above example, the strategy may buy 30 shares total; your long position is 30 shares. However, the `Sell` instruction only closes out the 10 shares bought using the `MyBuy` entry order. It ignores any other order, and does not close out the other 20 shares. Therefore, this strategy leaves you long 20 shares.

You can also close part of an entry order. For example, if your entry, which you named "`MyBuy`" buys 10 shares, you can specify that you want to exit from entry "`MyBuy`" but only close out 5 shares, not the entire 10:

```
Sell From Entry ("MyBuy") 5 Shares Next Bar at High + 3 Points Stop;
```

Important The entry name is case sensitive. Be sure to use consistent capitalization. Also, it is important to remember that exit orders do not pyramid; therefore, if an exit does not close out an entire position, you will need another exit order (or reversal order) in order to close out the position.

Number of Shares/Contracts

To specify how many shares/contracts to close out, use a numeric expression followed by the word `shares` or `contracts` after the trading verb `Sell`. Some Examples:

```
Sell 100 Shares This Bar on Close;

Sell From Entry ("MovAvg") 10 Shares Next Bar at High + 1 Point Stop ;
```

Note The words `shares` and `contracts` are synonymous.

If you do not specify the number of shares or contracts in the `Sell` instruction, the exit order closes out the entire long position, rendering your position flat.

When you specify the number of shares/contracts, the `Sell` instruction exits the specified number of shares/contracts from every open entry.

Therefore, if the Strategy allows for pyramiding, and has bought 500 shares twice (for a total of 1,000 shares), and an order to `Sell 100 Shares` is placed by the Strategy, the instruction will exit a total of 200 shares: 100 shares from each of the two entries.

However, if you want to exit a total of 100 shares, you can use the word `Total` in the `Sell` instruction. Using the word `Total` instructs the Strategy to exit 100 shares from the first open entry (first in, first out). For Examples:

```
Sell 100 Shares Total This Bar on Close;
Sell From Entry ("MovAvg") 10 Shares Total Next Bar at High + 1 Point Stop ;
```

Order Action

You must specify one of the five different order actions with your `Sell` order.

```
... this bar on close;
... next bar at open;
... next bar at market;
... next bar at yourprice Stop;
... next bar at yourprice Limit;
```

The execution method `this bar on close` is provided for backtesting purposes only; it en ables you to back test 'market at close' orders, which you cannot automate using TradeStation. Given that all orders are evaluated and executed at the end of each bar, TradeStation reads and issues the `this bar on close` order once the bar has closed (for example, once the daily trading session has ended). TradeStation fills the order using the close of the current bar, but you have to place an order at market to be executed on the next bar. This invariably introduces slippage.

The execution method `Sell next bar at` *`price`* `Limit` instructs TradeStation to exit a long position at the first opportunity at the specified *`price`* or higher. The execution method `Sell next bar at` *`price`* `Stop` instructs TradeStation to exit a long position at the first opportunity at the specified price or lower.

Stop and limit orders are treated by TradeStation as market if touched (MIT) orders. A MIT order becomes a market order when the price of the traded symbol reaches the specified target price. It is possible for stop and limit orders not to be filled (that is, price never reached); in this case, the orders are canceled at the close of the bar.

Tying the Exit Price to the Bar of Entry

When specifying the execution method, you can vary stop and limit orders by using '`At$`' instead of '`at`'. Using `At$` forces the strategy to refer to the value the numerical ex pression *`price`* had on the bar where the entry order was generated. Consider the following statement:

```
Sell From Entry ("MyBuy") Next Bar At$ Low - 1 Point Stop;
```

The above statement places an order to exit the long position at one point lower than the low of the bar where the order to establish the long position was generated (for example, if an order to `Buy next bar...` is generated today, the prices referenced will be today's, not tomorrow's. Even though the order was placed and filled tomorrow, it was generated today, and that is the bar referenced).

To use the `At$` reserved word, you must name the entry order, and the `Sell` instruction must refer to the specific entry order.

As another example, if the maximum risk you will tolerate for a position is 5 points under the closing price of the bar on which you generated the entry order, you can use the follow ing statement:

```
Sell From Entry ("MyBuy") Next Bar At$ Close - 5 Points Stop;
```

This is a valuable technique that allows you to refer easily to the prices of the bar on which the entry order was generated.

Example

This statement exits all contracts/shares of your open long position at the close of the current bar. Your position will be flat.

```
Sell This Bar on Close;
```

Additional Examples

The next instruction exits all contracts/shares of your positions opened by the entry order `Entry#1` at the open of the next bar, and the exit order is named `LongExit`.

```
Sell ("LongExit") From Entry ("Entry#1") Next Bar at Market;
```

The following statement places an order to close 5 contracts/shares in total at the low of the current bar minus 1 point or anything lower. This order is active throughout the next bar (until filled or canceled):

```
Sell 5 Contracts Total Next Bar at Low - 1 Point Stop;
```

The next instruction places an order to exit 100 shares from every entry at the high plus the range of the current bar or anything higher. This order is active throughout the next bar (until filled or canceled) and will be named `HighExit`.

```
Sell ("HighExit") 100 Shares Next Bar at High + Range Limit;
```

The following statement allows you to monitor your risk by placing an exit order 5 points below the closing price of the bar that generated the long entry order:

```
Sell From Entry ("MyBuy") Next Bar At$ Close - 5 Points Stop;
```

SellShort (Reserved Word)

Disclaimer

This trading verb is used to open a short position. A short position is created by covering all open long positions (if any) and opening a short position. The specifics of the order are defined by the optional parameters used in the statement (that is, number of shares, at what price, and so on).

```
SellShort [("Order Name")] [Number of Shares] Execution Method ;
```

Only the word `SellShort` and the `Execution Method` are required to open a short position. The following is a complete Easy Language statement:

```
SellShort This Bar on Close;
```

If no other parameters are specified, the default value for the `Order Name` is `"Short"`, and the number of contracts entered is then determined by the amount specified in the **Trade size** section of the **Strategy Properties for all Strategies** dialog box.

Each portion of the statement, `Order Name`, `Number of Shares`, and `Execution Method` is described next.

Order Name

When using multiple short entries within a strategy, it is helpful to label each entry order with a different name. By naming entry orders, you can easily identify all positions both on the chart and in the Strategy Performance Report. Also, naming the entry orders allows you to tie an exit to a particular entry. For more details, see `BuyToCover`.

To name a short entry order, include a descriptive name in quotation marks and within parenthesis after the trading verb `SellShort`. For example:

```
SellShort ("My Entry") This Bar on Close;
```

This instruction initiates a short position named `My Entry`. Again, when a strategy that contains this statement is applied to a price chart, the order name is displayed on the chart and in the Strategy Performance Report under the **Trades** tab.

Number of Shares/Contracts

To specify how many shares (or contracts) to open the short position with, place a numeric expression followed by the words `shares` (or `contracts`) after the trading verb `SellShort` (and entry order name if used). Some examples:

```
SellShort ("My Entry") 100 Shares Next Bar at Market;

SellShort 5 Contracts This Bar on Close;

SellShort Value1 Shares Next Bar at Market;
```

Note The words `shares` and `contracts` are synonymous.

If the number of shares/contracts is not specified, the value entered under the **Trade size** section of the **Strategy Properties for all Strategies** dialog box is used. The **Trade size** section controls the default trade amount; this can be set to either a fixed unit or dollars per transaction. The Trading Strategy Engine uses what ever is specified in this section only if the `SellShort` statement does not specify the number of shares/contracts with which to open the position.

Order Action

You must specify one of the five different order actions with your `SellShort` order.

```
... this bar on close;

... next bar at open;

... next bar at market;

... next bar at yourprice Stop;

... next bar at yourprice Limit;
```

The execution method `this bar on close` is provided for backtesting purposes only; it enables you to back test ‘market at close’ orders, which you cannot automate using TradeStation. Given that all orders are evaluated and executed at the end of each bar, TradeStation reads and issues the `this bar on close` order once the bar has closed (for example, once the daily trading session has ended). TradeStation fills the order using the close of the current bar, but you must place an order at market for execution on the next bar. This invariably introduces slippage.

An order to `SellShort next bar at` *`price`* `Limit` instructs TradeStation to sell at the first opportunity at the specified price or higher. A `SellShort next bar at` *`price`* `Stop` order instructs TradeStation to sell at the first opportunity at the specified price or lower.

Stop and limit orders are treated by TradeStation as market if touched (MIT) orders. A MIT order becomes a market order when the price of the symbol meets the specified order price. It is possible for stop and limit orders not to be filled (that is, price never met); in this case, the orders are canceled at the close of the bar.

Example

The following statement sells short 100 contracts/shares at the closing price of the current bar:

```
SellShort 100 Shares This Bar on Close;
```

Additional Examples

This statement sells short the default number of contracts/shares specified in the **Trade size** section of the **Strategy Properties for all Strategies** dialog box at the open of the next bar, and names this entry order `Entry#2`*:*

```
SellShort ("Entry#2") Next Bar at Market;
```

The next statement places an order to sell 5 contracts at the low of the current bar minus the range of the current bar, or any price lower. Note that `Range` is an EasyLanguage function that returns the high minus the low. This order remains active throughout the next bar (until filled or canceled).

```
SellShort 5 Contracts Next Bar at High + Range Stop;
```

The following statement places an order to sell 100 shares at the highest high of the last 10 bars, or any price higher. This order remains active throughout the next bar (until filled or canceled) and is named `HighSell`*:*

```
SellShort ("HighSell") 100 Shares Next Bar at Highest(High, 10) Limit ;
```

SERIESCOUNT (Reserved Word)

Disclaimer

SERIESCOUNT is an OptionStation reserved word that returns the number of series months in the option(n) array.

Sess1EndTime (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

Refer to SessionStartTime, SessionStartDay, SessionEndTIme, SessionEndDay for accessing session values.

Sess1FirstBarTime (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

Refer to the SessionFirstBarTIme function for accessing session bar values.

Sess1StartTime (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

Refer to SessionStartTime, SessionStartDay, SessionEndTIme, SessionEndDay for accessing session values.

Sess2EndTime (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

Refer to SessionStartTime, SessionStartDay, SessionEndTIme, SessionEndDay for accessing session values.

Sess2FirstBarTime (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

Refer to the SessionFirstBarTIme function for accessing session bar values.

Sess2StartTime (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

Refer to SessionStartTime, SessionStartDay, SessionEndTIme, SessionEndDay for accessing session values.

SessionCount (Reserved Word)

Disclaimer

Used with session information to refer to the number of sessions for the week. The number of session per week will vary based on the pre and post market settings in the current chart.

Usage

```
SessionCount (SessionType);
```

Parameters

`SessionType`	A numeric value. 0 = Auto Detect, 1 = Regular Session

Remarks

SessionType – Specifies the type of session information that should be returned. The type parameter may be specified as follows:

- Auto-Detect Session – [0, AutoSession] - uses whatever session is specified for the data series; regular or custom
- Regular Session – [1, RegularSession] - regular session information should always be returned

For Example: There are 5 regular sessions in a week for a stock.

Examples

Assign to Value1 the total number of sessions for the week in the current chart.

```
Value1 = SessionCount (0);
```

Asssign to Value1 the total number of regular session for the week in the current chart.

```
Value1 = SessionCount (1);
```

SessionCountMS (Reserved Word)

Disclaimer

Used with session information to refer to the number of sessions for the week, based on the merged sessions in multi-data charts.

MS - Merged sessions are from the earliest start time of all symbols to the latest end time for all sessions each trading day.

Usage

```
SessionCountMS;
```

Parameters

None

Remarks

For Example: There are generally 5 merged sessions in a week for a multi-data chart of stocks.

Session numbers are specified on the Format Symbols Properties dialog.

Examples

Assign to Value1 the total number of merged sessions for the week in the current chart.

```
Value1 = SessionCountMS;
```

SessionEndDay (Reserved Word)

Disclaimer

Used with session information to refer to the day that the specified session ends. Returns a numeric value for the day of the week the specified session ends: (0 = Sunday, 1 = Monday, etc.)

Usage

```
SessionEndDay(SessionType, SessionNum);
```

Parameters

`SessionType`	A numeric value. 0 = Auto Detect, 1 = Regular Session
`SessionNum`	A numeric value. Session Number to Reference

Remarks

SessionType – Specifies the type of session information that should be returned. The type parameter may be specified as follows:

- Auto-Detect Session – [0, AutoSession] - uses whatever session is specified for the data series; regular or custom
- Regular Session – [1, RegularSession] - regular session information should always be returned

SessionNum – Specifies the specific session that should be returned.

This is value ranges from 1 to the maximum number of sessions. Session numbers are available on the Format Symbols Properties dialog.

Examples

Assign to Value1 the end day for the 3rd regular session.

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

SessionEndDayMS (Reserved Word)

Disclaimer

Used with session information to refer to the day that the specified merged session ends in a multi-data chart. Returns a numeric value for the day of the week the specified merged session ends: (0 = Sunday, 1 = Monday, etc.)

MS - Merged sessions are from the earliest start time of all symbols to the latest end time for all sessions each trading day.

Usage

```
SessionEndDayMS(SsionNum);
```

Parameters

`SessionNum`	A numeric value. Session Number to Reference

Remarks

SessionNum – Specifies the specific session that should be returned.

This value ranges from 1 to the maximum number of sessions. Session numbers are available on the Format Symbols Properties dialog.

Examples

Assign to Value1 the end day for the 3rd merged session.

```
Value1 = SessionEndDayMS(3);
```

SessionEndTime (Reserved Word)

Disclaimer

Used with session information to refer to the time that the specified session ends. Returns a numeric value for the time of the day the specified session ends: (1600 = 4:00pm)

Usage

```
SessionEndTime(SessionType, SessionNum)
```

Parameters

SessionType	A numeric value. 0 = Auto Detect, 1 = Regular Session
SessionNum	A numeric value. Session Number to Reference

Remarks

SessionType – Specifies the type of session information that should be returned. The type parameter may be specified as follows:

- Auto-Detect Session – [0, AutoSession] - uses whatever session is specified for the data series; regular or custom
- Regular Session – [1, RegularSession] - regular session information should always be returned

SessionNum – Specifies the specific session that should be returned.

This value ranges from 1 to the maximum number of sessions. Session numbers are available on the Format Symbols Properties dialog.

Examples

Assign to Value1 the end time for the 3rd regular session.

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

SessionEndTimeMS (Reserved Word)

Disclaimer

Used with session information to refer to the time that the specified merged session ends in a multi-data chart. Returns a numeric value for the time of day the specified merged session ends: : (1600 = 4:00pm)

MS - Merged sessions are from the earliest start time of all symbols to the latest end time for all sessions each trading day.

Usage

```
SessionEndTimeMS(SessionNum);
```

Parameters

SessionNum	A numeric value. Session Number to Reference

Remarks

SessionNum – Specifies the specific session that should be returned.

This value ranges from 1 to the maximum number of sessions. Session numbers are available on the Format Symbols Properties dialog.

Examples

Assign to Value1 the end time for the 3rd merged session.

```
Value1 = SessionEndTimeMS(3);
```

SessionStartDay (Reserved Word)

Disclaimer

Used with session information to refer to the day that the specified session starts. Returns a numeric value for the day of the week the specified session starts: (0 = Sunday, 1 = Monday, etc.)

Usage

```
SessionStartDay(SessionType, SessionNum);
```

Parameters

`SessionType`	A numeric value. 0 = Auto Detect, 1 = Regular Session
`SessionNum`	A numeric value. Session Number to Reference

Remarks

SessionType – Specifies the type of session information that should be returned. The type parameter may be specified as follows:

- Auto-Detect Session – [0, AutoSession] - uses whatever session is specified for the data series; regular or custom
- Regular Session – [1, RegularSession] - regular session information should always be returned

SessionNum – Specifies the specific session that should be returned.

This is value ranges from 1 to the maximum number of sessions. Session numbers are available on the Format Symbols Properties dialog.

Examples

Assign to Value1 the start day for the 3rd regular session.

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

SessionStartDayMS (Reserved Word)

Disclaimer

Used with session information to refer to the day that the specified merged session starts in a multi-data chart. Returns a numeric value for the day of the week the specified merged session starts: (0 = Sunday, 1 = Monday, etc.)

MS - Merged sessions are from the earliest start time of all symbols to the latest end time for all sessions each trading day.

Usage

```
SessionStartDayMS(Session);
```

Parameters

`SessionNum`	A numeric value. Session Number to Reference

Remarks

SessionNum – Specifies the specific session that should be returned.

This value ranges from 1 to the maximum number of sessions. Session numbers are available on the Format Symbols Properties dialog.

Examples

Assign to Value1 the start day for the 3rd merged session.

```
Value1 = SessionStartDayMS(3);
```

SessionStartTime (Reserved Word)

Disclaimer

Used with session information to refer to the time that the specified session starts. Returns a numeric value for the time of the day the specified session starts: (930 = 9:30am)

Usage

```
SessionStartTime(SessionType, SessionNum)
```

Parameters

`SessionType`	A numeric value. 0 = Auto Detect, 1 = Regular Session
`SessionNum`	A numeric value. Session Number to Reference

Remarks

SessionType – Specifies the type of session information that should be returned. The type parameter may be specified as follows:

- Auto-Detect Session – [0, AutoSession] - uses whatever session is specified for the data series; regular or custom
- Regular Session – [1, RegularSession] - regular session information should always be returned

SessionNum – Specifies the specific session that should be returned.

This value ranges from 1 to the maximum number of sessions. Session numbers are available on the Format Symbols Properties dialog.

Examples

Assign to Value1 the start time for the 3rd regular session.

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

SessionStartTimeMS (Reserved Word)

Disclaimer

Used with session information to refer to the time that the specified merged session starts in a multi-data chart. Returns a numeric value for the time of day the specified merged session starts: (930 = 9:30am)

MS - Merged sessions are from the earliest start time of all symbols to the latest end time for all sessions each trading day.

Usage

```
SessionStartTimeMS( SessionNum);
```

Parameters

`SessionNum`	A numeric value. Session Number to Reference

Remarks

SessionNum – Specifies the specific session that should be returned.

This value ranges from 1 to the maximum number of sessions. Session numbers are available on the Format Symbols Properties dialog.

Examples

Assign to Value1 the start time for the 3rd merged session.

```
Value1 = SessionStartTimeMS(3);
```

SetAllOrNone (Reserved Word)

Disclaimer

Enables or disables the “All or None” advanced order setting.

No validation of orders will be done. Any incorrect combination will result in the trade server rejecting the order.

Usage:

SetAllOrNone(Flag)

Return:

None

Inputs:

Flag is a Boolean. Passing TRUE will enable the setting and passing FALSE will disable the setting.

Example:

A limit order to buy at the inside bid with the All or None advanced order enabled:

```
if lastbaronchart then begin
      SetAllOrNone(TRUE);
      Buy next bar at InsideBid Limit;
End;
```

SetBreakEven (Reserved Word)

Disclaimer

SetBreakEven is a built-in stop reserved word that allows you to exit a position at the breakeven point, once a specified profit floor has been reached. The profit floor can be specified on a total position basis, or a one contract or one share basis. Whether the profit target amount is based on a position or per share/contract basis is determined by set stop reserved words: SetStopPosition, SetStopShares, or SetStopContracts SetBreakEven does not factor in commissions or slippage.

SetBreakEven can only be used in a Strategy.

Usage

```
SetBreakEven(FloorAmt)
```

Input Parameters

Name	Type	Description
FloorAmt	Numeric	The profit amount (in dollars) that must be exceeded before the breakeven exit is activated.

Orders Generated

Market Order to Sell (Long Exit) or Market Order to Buy to Cover (Short Exit)

Signal Name

BreakEven Stop

Example

Exit all shares or contracts in all positions at the breakeven point if the position has been up at least $25.00 in profit at some point.

```
SetBreakEven(25)
```

Exit all shares or contracts in all positions at the breakeven point if the position has been up at least $0.25 from the entry price..

```
SetStopShare;
SetBreakEven(.25)
```

See Also

BreakEven Stop (strategy), SetStopPosition , SetStopShare , SetStopContract

SetBuyMinusSellPlus (Reserved Word)

Disclaimer

Enables or disables the "Buy on -/Sell on +" advanced order setting. This feature is only for listed orders.

No validation of orders will be done. Any incorrect combination will result in the trade server rejecting the order.

Usage:

SetBuyMinusSellPlus(Flag)

Return:

None

Inputs:

Flag is a Boolean. Passing TRUE will enable the setting and passing FALSE will disable the setting.

Example:

A market order to buy a NYSE stock at market on the next down-tick::

```
if lastbaronchart then begin
      SetBuyMinusSellPlus(TRUE);
      Buy next bar at Market;
End;
```

SetDiscretion (Reserved Word)

Disclaimer

Enables or disables the “Discretionary” advanced order setting.

No validation of orders will be done. Any incorrect combination will result in the trade server rejecting the order.

Usage:

SetDiscretion(DiscretionAMT)

Return:

None

Inputs:

DiscretionAMT is a double value. If greater than zero the “Discretionary” advanced setting is enabled using the value entered as the discretion to be added to the limit price. If zero the “Discretionary” advanced setting is disabled.

Example:

A limit order to buy that reflects in the market at .05 below the current insidebid, but will execute up to .05 cents above the current inside bid.

```
if lastbaronchart then begin
      SetDiscretion(.10);
      Buy next bar at InsideBid -.05 Limit; ;
end;
```

SetDollarTrailing (Reserved Word)

Disclaimer

This built-in stop reserved word is used to specify the amount, based on the maximum open position profit, you are willing to lose (in dollars). The position or contract/share is closed out when the specified amount is lost.

```
SetDollarTrailing(Amount)
```

`Amount` is the amount of the maximum open profit that you are willing to lose.

Note Use with `SetStopContract` or `SetStopPosition`.

Strategy

Dollar Risk Trailing

Example

To place a dollar risk trailing stop at $500 for the entire position, write:

```
SetStopPosition;
SetDollarTrailing(500);
```

As the price rises, so does the placement of the stop. It is maintained at a dollar value that results in a total of $500 loss for the entire position.

SetExitOnClose (Reserved Word)

Disclaimer

SetExitOnClose is a built-in stop reserved word used to place an order to exit all shares or contracts in all positions on the close of the last bar of the trading session on an intra-day chart.

For historical simulations, **SetExitOnClose** generates a market order on the bar close event of the last intra-day bar for each day in the chart. When used in an automated strategy placing real-world orders, **SetExitOnClose** generates a limit order into the post market trading session, (if one exists), otherwise a market order is generated for the open of the next regular session day.

SetExitOnClose by default uses the closing session time specified by the custom session settings of the chart.

SetExitOnClose can only be used in a Strategy.

Usage

```
SetExitOnClose
```

No Parameters

Note

When automating a strategy, you will not want to use **SetExitOnClose** as a way to make sure that all of your positions are closed prior to the regular session end time. To do this, you will need to generate a closing order prior to the last bar in the chart.

Orders Generated

Market Order to Sell (Long Exit) or Market Order to Buy to Cover (Short Exit)

or

Limit Order to Sell (Long Exit) or Limit Order to Buy to Cover (Short Exit) (automated strategy with a post market session)

Signal Name

End of Day Exit

Example

Exit all shares or contracts in all positions on the last bar of the trading day on an intra-day chart.

```
SetExitOnClose;
```

See Also

Close at End of Day (strategy)

SetFPCompareAccuracy (Reserved Word)

Disclaimer

SetFPCompareAccuracy will allow you to override the default precision tolerance value, allowing for a lower or higher tolerance when comparing two floating point values to be equal.

Two floating point values will be considered equal if “abs(Value1 – Value2) <= ε” where ε is the tolerance value. The scope of this function will be limited to the Indicator, Strategy, or Function in which it was called.

Versions of TradeStation prior to 7.0 used 4-byte single precision floating point values for variables and a less precise tolerance, approximately 5.0e-6, when comparing two variables for equality. Starting with Version 7.0 EasyLanguage uses 8-byte double precision floating point values and a very high tolerance of 2.2204460492503131e-016. This difference in tolerance checking may cause indicator values between version 7.0 and prior versions to be slightly different. Specifically, two values that were considered equal in versions prior to 7.0 are no longer considered equal in version 7.0. In order to address this issue we have changed the default tolerance value to 2.2204460492503131e-012.

Usage:

SetFPCompareAccuracy(AccuracyType)

Return:

None

Inputs:

AccuracyType is an integer constant and may be one of the following values:

These a re the EasyLanguage constants and their values to pass into SetFPCompareAccuracy, they are defined as follows:

0 = fpcVeryLowAccuracy - tolerance to: 2.2204460492503131e-08

1 = fpcLowAccuracy - tolerance to: 2.2204460492503131e-010

2 = fpcMedAccuracy - tolerance to: 2.2204460492503131e-012 (this is the default value)

3 = fpcHighAccuracy - tolerance to: 2.2204460492503131e-014

4 = fpcVeryHighAccuracy - tolerance to: 2.2204460492503131e-016

5 = fpcExact - tolerance to: 0.0

Example:

Set floating point accuracy to very high tolerance accuracy:

```
SetFPCompareAccuracy(fpcVeryHighAccuracy );
```

SetNonDisplay (Reserved Word)

Disclaimer

Enables or disables the “Non-Display” advanced order setting.

No validation of orders will be done. Any incorrect combination will result in the trade server rejecting the order.

Usage:

SetNonDisplay(Flag)

Return:

None

Inputs:

Flag is a Boolean. Passing TRUE will enable the setting and passing FALSE will disable the setting.

Example:

A limit order to buy at the inside bid with the Non Display advanced order enabled:

```
if lastbaronchart then begin
      SetNonDisplay(TRUE);
      Buy next bar at InsideBid Limit;
End;
```

SetNOW (Reserved Word)

Disclaimer

Enables or disables the “NOW (ECN’s Only)” advanced order setting.

No validation of orders will be done. Any incorrect combination will result in the trade server rejecting the order.

Usage:

SetNOW(Flag)

Return:

None

Inputs:

Flag is a Boolean. Passing TRUE will enable the setting and passing FALSE will disable the setting.

Example:

A limit order to buy at the inside bid with the NOW advanced order enabled:

```
if lastbaronchart then begin
      SetNOW(TRUE);
      Buy next bar at InsideBid Limit;
End;
```

SetPeg (Reserved Word)

Disclaimer

Enables or disables the “Peg” advanced order feature.

No validation of orders will be done. Any incorrect combination will result in the trade server rejecting the order.

Usage:

SetPeg(PegValue)

Return:

None

Inputs:

PegValue may be one of the following constants: pegBest, pegMid, pegDisabled. If pegDisabled is passed the “Peg” setting is disabled otherwise “Peg” is enabled.

These a re the EasyLanguage constants and their values to pass into SetPeg, they are defined as follows:

0 = PegDisable (Disable Peg Feature)

1 = PegBest (Best Bid for a Buy, Best Ask for a Sell Short)

2 = PegMid (Mid Price for both Buy and Sell Short)

Example:

A limit order to buy and Peg at best bid up to the InsideBid plus .10:

```
if lastbaronchart then begin
      SetPeg(pegBest);
      Buy next bar at InsideBid +.10 Limit;
end;
```

A limit order to buy and Peg at mid price up to the InsideBid plus .10:

```
if lastbaronchart then begin
      SetPeg(pegMid);
      Buy next bar at InsideBid +.10 Limit;
end;
```

SetPercentTrailing (Reserved Word)

Disclaimer

This is a built-in stop reserved word is used to specify the amount of the maximum open position profit you are willing to lose (as a percent) as well as the profit level that must be reached in order for the stop to take effect. The position or contract/share is closed out when the specified percentage of the maximum profit is lost.

```
SetPercentTrailing(FloorAmnt, Amount)
```

`FloorAmnt` is a numeric expression representing the amount of profit to be reached before the stop takes effect.
`Amount` is the percent of the profit you are willing to lose.

Note Use with `SetStopContract` or `SetStopPosition`.

Strategy

PercentRisk Trailing

Example

In order to place a percentage based trailing stop that exits a position once it has returned 15% of the maximum equity earned after the position has made $500, you would write:

```
SetStopPosition;
SetPercentTrailing(500, 15);
```

SetPlotBGColor (Reserved Word)

Disclaimer

This reserved word is used to change the background plot color of a particular RadarScreen cell.

```
SetPlotBGColor(Number, Color);
```

`Number` is a number from 1 to 99 representing the number of the plot to modify. `Color` is the EasyLanguage color to be used for the plot.

Example

The following EasyLanguage statements color a cell background plot green when *Condition1* is true ('bullish"0), and red when it is false ("bearish).:

```
If Condition1 then begin
 Txt = "Bullish";
  SetPlotBGColor(1, green);
end

else begin
 Txt = "Bearish";
  SetPlotBGColor(1, red);

end;
```

SetPlotColor (Reserved Word)

Disclaimer

This reserved word is used to change the color of a particular plot in a price chart or grid.

```
SetPlotColor(Number, Color);
```

`Number` is a number from 1 to 99 representing the number of the plot to modify. `Color` is the EasyLanguage color to be used for the plot.

Example

The following EasyLanguage statements change the color of the plot to `Blue` when the RSI Indicator is over 75, and Green when it is under 25.

```
Plot1(RSI(Close, 9), "RSI") ;
If Plot1 > 75 Then
 SetPlotColor(1, Blue);
If Plot1 < 25 Then
 SetPlotColor(1, Green);
```

Notice that the initial `Plot1` statement resets the plot color to the indicator's default color setting each time the EasyLanguage code is run on each bar. This means that the indicator plots the default color unless the RSI value is greater than 75 or less than 25.

SetPlotType (Reserved Word)

Disclaimer

This reserved word sets the data type of the specified plot, typically in a RadarScreen cell. `SetPlotType` changes the type attribute and applies it to a given plot only once when the EasyLanguage code is evaluated.

```
SetPlotType(Number, DataType);
```

`Number` is a number from 1 to 99 representing the number of the plot to modify. `DataType` is the EasyLanguage data type to be used for the plot (see table below). For example, this allows you to change the plot display type of data in a RadarScreen cell (from Double to Date) .

DataType	
Name	**Value**
ptDate	25
ptDateTime	27
ptDouble	28
ptFloat	29
ptInvalid	3
ptProbability	4
ptString	2
ptTime	26
ptTrueFalse	1

SetPlotWidth (Reserved Word)

Disclaimer

This reserved word sets the width of the specified plot.

```
SetPlotWidth(Number, Width);
```

`Number` is a number from 1 to 99 representing the number of the plot to modify. `Width` is the EasyLanguage width to be used for the plot.

Example

The following EasyLanguage statements change the width of the plot to a thicker line when the Momentum Indicator is over 0, and to a thinner line when it is under 0:

```
Plot1(Momentum(Close, 10), "Momentum") ;

If Plot1 > 0 Then
 SetPlotWidth(1, 4);

If Plot1 < 0 Then
 SetPlotWidth(1, 1);
```

In this example, the *Momentum* Indicator has two possible widths: thicker when it is over 0, and thinner when it is below 0. However, in some cases you will want the indicator to have three or more possible widths.

SetProfitTarget (Reserved Word)

Disclaimer

SetProfitTarget is a built-in stop reserved word that enables you to specify the amount of profit you are looking to capture on a total position basis, or a one contract or one share basis. Within a strategy an order to close your entire position is generated once the profit target amount has been reached. Whether the profit target amount is based on a position or per share/contract basis is determined by set function reserved words: SetStopPosition, SetStopShare, or SetStopContracs

When used with an automated strategy where orders are being generated into the real-world, the profit exit limit order is immediately sent into the market to be filled. If you lose internet connectivity, your profit target is still active, but your strategy can longer generate additional orders.

Normally strategies generate orders on the close of the bar for execution on the next bar, **SetProfitTarget** allows you to generate orders and exit on the same bar as the bar of entry, this is especially useful when working with longer during bars, (e.g. 30-min, 60-min, daily, weekly, monthly).

SetProfitTarget can only be used in a Strategy.

Usage

```
SetProfitTarget(Amount)
```

Input Parameters

Name	Type	Description
Amount	Numeric	Amount of profit in dollars, at which point the position will be closed.

Examples

When used with the SetStopPosition reserved word, the Amount parameter sets the stop loss as a dollar amount for the entire position based on the total number of shares or contracts in the current open position. (if you held 500 shares of MSFT and specified $200 profit amount, you would exit $0.40 from your entry price.)

```
SetStopPosition;
SetProfitTarget(200);
```

(SetStopPosition is the default setting and is optional)

When used with the SetStopShare or SetStopContract reserved words, the Amount parameter sets the profit as a one share or one contract amount. (if you held 500 shares of MSFT and specified $0.60 profit amount, you would exit .60 from your entry price with a $300 profit.)

```
SetStopShare;
SetProfitTarget(.60);
```

(SetStopShare and SetStopContract are synonymous and both work for stock, futures, options, etc.)

Orders Generated

Limit Order to Sell (Long Exit) or Limit Order to Buy to Cover (Short Exit)

Signal Name

Profit Target

Related Functions

ProfitTarget (strategy) , StopLoss (strategy) , SetStopLoss , SetStopPosition , SetStopShare , SetStopContract

SetRoute (Reserved Word)

Disclaimer

Sets the order routing for the strategy order.

No validation of orders will be done. Any incorrect combination will result in the trade server rejecting the order.

Usage:

SetRoute(RouteName)

Return:

None

Inputs:

RouteName is a constant and may be one of the following values:

These a re the EasyLanguage constants for routes and their values to pass into SetRoute, they are defined as follows:

1 = rtIntelligent

2 = rtARCA

3 = rtBRUT

4 = rtBTRD

5 = rtINCA

6 = rtISLD

7 = rtSuperMont

8 = rtSuperDOT

Example:

A limit order to buy at the inside bid routed by Intelligent routing:

```
if lastbaronchart then begin
      SetRoute(rtIntelligent);
      Buy next bar at insidebid Limit;
end;
```

A limit order to buy at the inside bid routed to the Island ECN:

```
if lastbaronchart then begin
      SetRoute(rtISLD);
      Buy next bar at insidebid Limit;
end;
```

SetShaveImprove (Reserved Word)

Disclaimer

Adjusts the strategy calculated limit price by the user defined ShaveImproveVal.

No validation of orders will be done. Any incorrect combination will result in the trade server rejecting the order.

Usage:

SetShaveImprove(ShaveImproveVal)

Return:

None

Inputs:

ShaveImproveVal is a double value. If zero the strategy limit price is not adjusted. If greater than or less that zero the strategy limit price is added to ShaveImproveVal before being sent to the market.

Example:

A limit order to buy at the InsideBid and shave .01;

```
if lastbaronchart then begin
      SetShaveImprove(-.01);
      Buy next bar at InsideBid Limit;
end;
```

A limit order to buy at the InsideBid and Improve .01;

```
if lastbaronchart then begin
      SetShaveImprove(.01);
      Buy next bar at InsideBid Limit;
end;
```

SetShowOnly (Reserved Word)

Disclaimer

Enables or disables the “Show Only” advanced order setting. The total number of shares to buy is specified in either the strategy properties dialog or in the EasyLanguage code of the strategy.

No validation of orders will be done. Any incorrect combination will result in the trade server rejecting the order.

Usage:

SetShowOnly(Quantity)

Return:

None

Inputs:

Quantity is an Integer value. If greater than zero the “Show Only” advanced setting is enabled using the value entered as the number of shares to be reflected to the market. If zero the “Show Only” advanced setting is disabled.

Example:

A limit order to buy 1000 shares but reflecting only 200 in the market:

```
if lastbaronchart then begin
      SetShowOnly(200);
      Buy next bar 1000 shares at InsideBid Limit;
end;
```

SetStopContract (Reserved Word)

Disclaimer

SetStopContract is a reserved word that sets the way built-in stops calculate the exit order prices. Each built-in stop reserved word has a parameter for the amount of profit or loss desired, SetStopContract is simply a switch that changes the behavior of the built-in stops so that the exit order is calculated on a per contract/share basis, otherwise the exit order is calculated on a total position basis.

SetStopContract is synonymous with SetStopShare.

If SetStopContract, SetStopShare, and SetStopPosiiton are used multiple times, even in different strategies applied to the same chart, the last reference is controlling.

Example 1

When used with the SetStopLoss built-in exit strategy, SetStopContract sets the exit order based on a per contract basis.

If you held the S&P e-mini and set a $75 stop loss amount, you would exit 1.50 from your entry price with a $75 loss for each contract held. For example, 10 contracts would result in a $750 stop loss ($75 x 10 contracts)

```
SetStopContract;
SetStopLoss(75);
```

Example 2 - Forex

When used with a Forex symbol, such as EURUSD, a contract refers to a lot (typically consisting of 100,000 units).

```
Buy 1 contract next bar at market;
SetStopContract;
SetStopLoss(.001);
```

This example enters into a position of 1 lot size (100,000 units) and then attempts to exit based on a stop loss of .001 per unit, which for this Forex symbol would be a stop loss of $100 for the lot (.001 * 100,000 units).

Related Functions

SetStopPosition , SetStopShare

SetStopLoss (Reserved Word)

Disclaimer

SetStopLoss is a built-in stop reserved word that enables you to specify the amount of money you are willing to risk either on a total position basis, or a one contract or one share basis. Within a strategy an order to close your entire position is generated once the stop loss amount has been reached. Whether the stop loss amount is based on a position or per share/contract basis is determined by set stop reserved words: SetStopPosition, SetStopShare, or SetStopContract

When used with an automated strategy where orders are being generated into the real-world, the stop exit order is monitored and the order held on your computer, once the Stop Loss amount is reached, a market order is generated and sent to be executed. If you lose internet connectivity, you no longer have a working stop.

Normally strategies generate orders on the close of the bar for execution on the next bar, **SetStopLoss** allows you to generate orders and exit on the same bar as the bar of entry, this is especially useful when working with longer during bars, (e.g. 30-min, 60-min, daily, weekly, monthly).

SetStopLoss can only be used in a Strategy.

Usage

```
SetStopLoss(Amount)
```

Input Parameters

Name	Type	Description
Amount	Numeric	Amount of loss, in dollars, at which point the position will be closed.

Examples

When used with the SetStopPosition reserved word, the Amount parameter sets the stop loss as a dollar amount for the entire position based on the total number of shares or contracts in the current open position. (if you held 500 shares of MSFT and specified $200 stop loss amount, you would exit $0.40 from your entry price.)

```
SetStopPosition;
SetStopLoss(200);
```

(SetStopPosition is the default setting and is optional)

When used with the SetStopShare or SetStopContract reserved words, the Amount parameter sets the stop loss as a one share or one contract amount. (if you held 500 shares of MSFT and specified $0.60 stop loss amount, you would exit .60 from your entry price with a $300 loss.)

```
SetStopShares;
SetStopLoss(.60);
```

(SetStopShare and SetStopContract are synonymous and both work for stock, futures, options, etc.)

Orders Generated

Market Order to Sell (Long Exit) or Market Order to Buy to Cover (Short Exit)

Signal Name

Stop Loss

Related Functions

StopLoss (strategy) , ProfitTarget (strategy) , SetProfitTarget , SetStopPosition , SetStopShare , SetStopContract

SetStopPosition (Reserved Word)

Disclaimer

SetStopPosiiton is a reserved word that sets the way built-in stops calculate the exit order prices. Each built-in stop command has a parameter for the amount of profit or loss desired, SetStopPosiiton is simply a switch that changes the behavior of the built-in stop command so that the exit order is calculated on a total position basis, verses a per share / per contract basis, set by the SetStopContract or SetStopShare switches**.**

SetStopPosiiton is the default setting for all buit-in set commands.

If SetStopShare, SetStopContract, and SetStopPosiiton are used multiple times, even in different strategies applied to the same chart, the last reference is controlling.

Example

When used with the SetStopLoss built-in exit strategy, SetStopContracts sets the exit order based on a per contract basis.

(if you held 1000 shares of the MSFT and set a $250 stop loss amount, you would exit $0.25 from your entry price with a $250 loss. ($250 / 1000 shares))

```
SetStopPosition;
SetStopLoss(250);
```

Related Functions

SetStopLoss, SetStopContract

SetStopShare (Reserved Word)

Disclaimer

SetStopShare is a reserved word that sets the way built-in stops calculate the exit order prices. Each built-in stop reserved word has a parameter for the amount of profit or loss desired, SetStopShare is simply a switch that changes the behavior of the built-in stops so that the exit order is calculated on a per share/contract basis, otherwise the exit order is calculated on a total position basis.

SetStopShare is synonymous with SetStopContract.

If SetStopShare, SetStopContract, and SetStopPosiiton are used multiple times, even in different strategies applied to the same chart, the last reference is controlling.

Example

When used with the SetStopLoss built-in exit strategy, SetStopContract sets the exit order based on a per contract basis.

(if you held 1000 shares of the MSFT and set a $0.25 stop loss amount, you would exit $0.25 from your entry price with a $250 loss. ($0.25 x 1000 shares))

```
SetStopShare;
SetStopLoss(.25);
```

Example 2 - Forex

When used with a Forex symbol, such as EURUSD, a share refers to a lot (typically consisting of 100,000 units).

```
Buy 1 share next bar at market;
SetStopShare;
SetStopLoss(.001);
```

This example enters into a position of 1 lot size (100,000 units) and then attempts to exit based on a stop loss of .001 per unit, which for this Forex symbol would be a stop loss of $100 for the lot (.001 * 100,000 units).

Related Functions

SetStopPosition , SetStopContract

SetSubscriberOnly (Reserved Word)

Disclaimer

Enables or disables the “Subscriber Only” advanced order setting.

No validation of orders will be done. Any incorrect combination will result in the trade server rejecting the order.

Usage:

SetSubscriberOnly(Flag)

Return:

None

Inputs:

Flag is a Boolean. Passing TRUE will enable the setting and passing FALSE will disable the setting.

Example:

A limit order to buy at the inside bid with the Subscriber Only advanced order enabled:

```
if lastbaronchart then begin
      SetSubscriberOnly(TRUE);
      Buy next bar at InsideBid Limit;
End;
```

Settlement (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

SGA_Exp_By_NetSales (Reserved Word)

Disclaimer

Retained for backward compatibility.

Share (Reserved Word)

Disclaimer

Reserved word used to specify the number of units to trade within a trading strategy.

```
   Share
```

Remarks

`Share` is normally used in an Entry or Exit statement.

Note With Forex symbols, 1 share refers to 1 lot (typically 100,000 units).

Examples

```
   Buy 1 share next bar at market
```

Generates an order to buy one share at the open of the next bar.

```
   SellShort 1 share next bar at market
```

Generates an order to sell one share at the open of the next bar.

Additional Example

To exit from only 1 contract from multiple long positions, write:

```
   Sell 1 share total next bar at market;
```

Shares (Reserved Word)

Disclaimer

Reserved word used to specify the number of units to trade within a trading strategy.

```
Shares
```

Remarks

`Shares` is normally used in an Entry or Exit statement.

Note With Forex symbols, 1 share refers to 1 lot (typically 100,000 units).

Examples

```
Buy 5 shares next bar at market
```

Generates an order to buy five shares at the open of the next bar.

```
SellShort 5 shares next bar at market
```

Generates an order to sell five shares at the open of the next bar.

Additional Example

To exit from only 5 contracts from multiple long positions, write:

```
Sell 5 shares total next bar at market;
```

SharesOut (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

Short (Reserved Word)

Disclaimer

Retained for backward compatibility.

ShortRollAdj (Reserved Word)

Disclaimer

Reserved word that returns an end-of-day roll adjustment factor for short forex positions.

```
ShortRollAdj
```

Remarks

ShortRollAdj is used to apply a short position roll adjustment for p/l calculations on forex short positions that are held overnight to account for fluctuations in daily interest rates.

Examples

Assigns to Value1 the short position roll adjustment factor for the current bar.

```
Value1 = ShortRollAdj;
```

Assigns to Value2 the short position roll adjustment factor from 7 bars ago.

```
Value2 = ShortRollAdj[7];
```

SICCode (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

Sign (Reserved Word)

Disclaimer

Returns an integer based on the sign on `Num`. 1 is returned for a positive value, -1 is returned for a negative value, and 0 is returned for zero.

`Sign(Num)`

`Num` is a numeric expression to be used in the calculation.

Examples

`Sign(56.23)` returns a value of 1.

`Sign(-9.5687)` returns a value of -1.

Sine (Reserved Word)

Disclaimer

Returns the sine value of the specified `Num` of degrees.

`Sine(Num)`

`Num` is a numeric expression to be used in the calculation.

Remarks

The sine is the trigonometric function that for an acute angle is the ratio between the leg opposite the angle when it is considered part of a right triangle and the hypotenuse.

`Num` should be the number of degrees in the angle.

Examples

`Sine(45)` returns a value of 0.7071.

`Sine(72)` returns a value of 0.9511.

Skip (Reserved Word)

Disclaimer

This word has been reserved for future use.

Slippage (Reserved Word)

Disclaimer

Returns the slippage setting of the applied strategy.

```
Slippage
```

Remarks

This reserved word can only be used in the evaluation of strategies.

Example

`Slippage` returns a value of .50 if the slippage has been set to .5.

SnapFundExists (Reserved Word)

Reserved for backward compatibility.

Spaces (Reserved Word)

Disclaimer

Adds the specified number of blank spaces into the line of commentary or text output.

```
Spaces(Cnt);
```

`Cnt` is the numeric expression indicating the number of spaces to be inserted.

Example

```
   Print("Close" + Spaces(5) + NumToStr(Close, 3));
```

The above example results in five blank spaces between the string "Close" and the closing price.

Square (Reserved Word)

Disclaimer

Returns the square of the specified `Num`.

`Square(Num)`

Where `Num` is a numeric expression to be used in the calculation.

Remarks

The square is a number raised to the 2nd power.

Examples

`Square(6.23)` returns a value of 38.8129.

`Square(-9.5687)` returns a value of 91.5600.

SquareRoot (Reserved Word)

Disclaimer

Returns the square root of the specified `Num`.

`SquareRoot(Num)`

Where `Num` is a numeric expression to be used in the calculation.

Remarks

The square root is the number that must be raised to the 2nd power in order to produce a given number, `Num`.

`Num` must be a positive number or zero.

Examples

`SquareRoot(6.23)` returns a value of 2.4960.

`SquareRoot(121)` returns a value of 11.

StartDate (Reserved Word)

Disclaimer

Reserved for future use.

stAutoSession (Reserved Word)

Disclaimer

Returns a numeric value of 0 that represents a session type of Auto.

```
stAutoSession;
```

Example

```
  Value1 = SeesionFirstBarTime(stAutoSession,3);
```

StockSplit (Reserved Word)

Disclaimer

Retained for backward compatibility.

StockSplitCount (Reserved Word)

Disclaimer

Retained for backward compatibility.

StockSplitDate (Reserved Word)

Disclaimer

Retained for backward compatibility.

StockSplitTime (Reserved Word)

Disclaimer

Retained for backward compatibility.

Stop (Reserved Word)

Disclaimer

Used in an entry or exit statement to specify how to fill an order.

```
Stop
```

Remarks

- Stop orders can only be executed on the next bar.
- Stop can be read as "this price or worse", meaning higher for a Long Entry and Short Exit, lower for a Short Entry and Long Exit.
- Stop orders require a reference price.

Examples

```
Buy next bar at 75 Stop;
```

Generates an order to enter a long position on the next bar at a price of 75 or higher.

```
SellShort next bar at 75 Stop;
```

Generates an order to enter a short position on the next bar at a price of 75 or lower.

Additional Example

```
Sell next bar at 75 Stop;
```

Generates an order to exit a long position on the next bar at a price of 75 or lower.

stRegularSession (Reserved Word)

Disclaimer

Returns a numeric value of 1 that represents a session type of Regular.

```
stRegularSession;
```

Example

```
   Value1 = SessionCountDay(stRegularSession,2);
```

STRIKE (Reserved Word)

Disclaimer

STRIKE is an OptionStation reserved word that returns the strike price of an option.

STRIKECOUNT (Reserved Word)

Disclaimer

STRIKECOUNT is an OptionStation reserved word, reserved for future use. It is the number of strikes available in the option(n) array.

STRIKEITMCOUNT (Reserved Word)

Disclaimer

Reserved for future use.

STRIKEOTCOUNT (Reserved Word)

Disclaimer

Reserved for future use.

String (Reserved Word)

Disclaimer

Reserved word used to define an input that expects a text string passed by value.

`String`

Remarks

`String` can be used for inputs that can be either `StringSimple` or `StringSeries`.

Examples

```
Input: MyMessage(String);
```

declares the constant `MyMessage` as a text string value to be used in a function.

```
Input: NewMessage(String);
```

declares the constant `NewMessage` as a text string value to be used in a function.

StringArray (Reserved Word)

Disclaimer

Reserved word used to define an input that expects a text string passed by value for each array element.

```
InputName(StringArray);
```

Remarks

A function input is declared as a `StringArray` when it is passing in a text string array by value.

Examples

```
   Input: PassedValues[n](StringArray);
```

indicates that a text string array is being passed into the function by value through the Input `PassedValues`.

StringArrayRef (Reserved Word)

Disclaimer

Reserved word used to define an input that expects a text string passed by reference.

```
InputName(StringArrayRef);
```

Remarks

A function input is declared as a `StringArrayRef` when it is passing in a text string array by reference.

Examples

```
   Input: PassedValues[n](StringArrayRef);
```

indicates that a text string array is being passed into the function by reference through the Input `PassedValues`.

StringRef (Reserved Word)

Disclaimer

Reserved word used to define an input that expects a text string passed by reference.

```
StringRef
```

Examples

```
   Input: MyMessage(StringRef);
```

declares the constant MyMessage as a String value to be used in a function.

```
   Input: NewMessage(StringRef);
```

declares the constant NewMessage as a String value to be used in a function.

StringSeries (Reserved Word)

Disclaimer

Reserved word used to define an input as a text series expression with history.

```
InputName(StringSeries);
```

Remarks

`StringSeries` is used for inputs whose values can be referred to historically.

Examples

```
   Input: MyMessage(StringSeries);
```

declares the constant `MyMessage` as a text string to be used in a function, where historical values of `MyMessage` are available.

```
   Input: NewMessage(StringSeries);
```

declares the constant `NewMessage` as a text string to be used in a function, where historical values of `NewMessage` are available.

StringSimple (Reserved Word)

Disclaimer

Reserved word used to define an input as a text simple expression without history available.

```
InputName(StringSimple);
```

Remarks

`StringSimple` can not be used for inputs whose values can be referred to historically.

Examples

```
   Input: MyMessage(StringSimple);
```

declares the constant `MyMessage` as a text string value to be used in a function, restricting `MyMessage` to be a value that does not contain historical values.

```
   Input: NewMessage(StringSimple);
```

declares the constant `NewMessage` as a text string value to be used in a function, restricting `NewMessage` to be a value that does not contain historical values.

StringToDate (Reserved_Word)

Disclaimer

This reserved word returns only the date portion of a DateTime value representing the specified string expression of the date in "MM/DD/YY" or "MM/DD/YYYY" format.

```
StringToDate(strValue);
```

Remarks

The specific date must be a valid date between January 1, 1900 and February 28, 2150, expressed in DateTime format.

Examples

`Value1 = StringToDate("02/01/03");` returns a DateTime value of 37653 for the specified date of Feb. 1 2003.

StringToDateTime (Reserved_Word)

Disclaimer

This reserved word returns a double-precision decimal DateTime value representing the specified string expression of the date and time in "MM/DD/YYYY HH:MM:SS TT" format.

```
StringToDateTime(strValue);
```

Remarks

The specific date must be a valid date between January 1, 1900 and February 28, 2150, expressed in DateTime format.

Examples

`Value1 = StringToDateTime("02/01/03 10:12:43 PM");` returns a DateTime value of 37653.93 for the specified date and time.

StringToTime (Reserved_Word)

Disclaimer

This reserved word returns only the time portion of a DateTime value representing the specified string expression of the date in "HH:MM:SS TT" format.

```
StringToTime(strValue);
```

Remarks

The specific time must be a valid time.

Examples

```
Value1 = StringToTime("10:12:43 PM");
```

returns a DateTime value of 0.93 for the specified date of 10:12:43 PM.

StrLen (Reserved Word)

Disclaimer

Counts and returns number of characters in the specified string expression.

```
StrLen("Str");
```

`Str` is the string expression to count. It must be enclosed in quotation marks.

Example

`StrLen("Net Profit")` returns the numeric expression 10 for the number of characters in the string.

StrToNum (Reserved Word)

Disclaimer

Converts the specified string expression to a numeric value.

```
StrToNum("Str");
```

`Str` is the string expression to convert to a numeric expression. It must be enclosed in quotation marks.

Remarks

If any non-numeric characters are included in the string expression, zero (0) is returned. The only exception is when non-numeric characters are located at the end of the string expression, in which case, they are dropped from the numeric expression.

Example

`StrToNum("1170.50")` returns the numeric expression 1170.50.

SumList (Reserved Word)

Disclaimer

Returns the sum of the inputs.

```
SumList(Num1, Num2, Num3, etc.)
```

`Num1, Num2,` and `Num3` are numeric expressions representing the values to be used in the calculation.

Examples

`SumList(45, 72, 86, 125, 47)` returns a value of 375.

`SumList(18, 67, 98, 24, 65, 19)` returns a value of 291.

Sunday (Reserved Word)

Disclaimer

Returns the number 0 as the value for Sunday.

```
Sunday
```

Example

```
   Sunday
```

…returns a value of 0 as the value for Sunday.

Switch/Case (Reserved Word)

Disclaimer

Reserved word used to transfer control to an associated Case or Default statement.

Syntax

```
Switch ( expression )
Begin
Case case-expression:  statements;
Default: statements;
End;
```

Control passes to the statements whose case-expression matches the value of the switch (expression). The switch statement can include any number of case instances, but no two case constants within the same switch statement can have the same constant value.

Note Once the statements associated with a matching case are evaluated, control passes to the end of the switch statement. This is an implied break and is different than a similar structure found in some other languages that require an explicit break.

Remarks

A single case statement can carry multiple values, as the following example shows:

```
  Case 1, 2, 3, 4, 5, 6, 20: Value1 = Lowest(Close,3);
```

Ranges like this are also valid:

```
  Case 1 to 6, 20: Value2 = Highest(High,5);
```

In both of the above examples, if case-expression equals any number between 1 and 6 or equal to 20, a function is called and assigned to a value.

In addition, logical operators may be used with case statements including: >, <, >=, <=, <> and =.

```
  Case > Average( Close, 50 ): Value1 = Close ;
```

The “is” keyword is an EasyLanguage skip keyword and can be use for better clarity as in the following:

```
  Case is < Average( High, 75 ): Value1 = High ;
```

Example

```
   Switch(Value1)
   Begin
     Case 1 to 5:
        Value2 = Value2 + 1;

       Case 10, 20, 30:

           Value3 = Highest(High,10);

       Case is > 40:

           Value3 = Value3 + 1;

       Default:
        Value5 = Value5 + 1;
   End;
```

The above switch statement Increments `Value2` when the switch expression (`Value1`) is a value from 1 to 5; assigns `Value3` to the highest high of 10 bars ago when the switch expression is either 10, 20 or 30; increments `Value4` for all switch expression values above 40; and increments `Value5` for any other values of the switch expression.

Symbol (Reserved Word)

Disclaimer

A quote field that returns a string expression representing the symbol name. See also GetSymbolName.

SymbolName (Reserved Word)

Disclaimer

A quote field that returns the symbol name.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

SymbolRoot (Reserved Word)

Disclaimer

A quote field that returns a string expression containing the symbol root.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

Remarks

Returns a string expression containing the root of the symbol based on the symbol type as follows:

- **Futures** - The future root symbol minus any month, year, or extension (such as .C, .D, .P)
- **Stock and Index Options** - The option root symbol that includes the leftmost portion of the complete stock or index option symbol prior to the space.
- **Future Options** - The option root symbol will always be a complete future symbol including the month, year, and extension (such as .C, .D, .P) if applicable.
- **All Other** - The complete symbol will be returned.

T (Reserved Word)

Disclaimer

This reserved word returns a numeric expression representing the EasyLanguage time (HHMM format) of the closing price of the current bar. Abbreviation of the reserved word `Time`.

```
T
```

Remarks

Anytime the Time of a bar is needed, the letter `T` can be used in an equivalent fashion.

`T` returns the time in 24-hour format.

Examples

`T` returns 1600 if the Time of the bar is 4:00pm.

`T` returns 0930 if the Time of the bar is 9:30am.

Additional Example

To check the Time of the previous bar write the following code:

```
   T of 1 bar ago
```

Tangent (Reserved Word)

Disclaimer

Returns the tangent value of the specified Num of degrees.

```
Tangent(Num)
```

Num is a numeric expression to be used in the calculation.

Remarks

The tangent the trigonometric function that for an acute angle is the ratio between the leg opposite to the angle when it is considered part of a right triangle and the leg adjacent, also equal to the sine divided by the cosine.

Num should be the number of degrees in the angle.

Examples

Tangent(45) returns a value of 1.0.

Tangent(72) returns a value of 3.0776.

TARGET (Reserved Word)

Disclaimer

Reserved for future use.

TARGETTYPE (Reserved Word)

Disclaimer

TICKTYPE is an OptionStation reserve word that returns the type of values to be calculated in the current core calculation event.

Text (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

Text_Delete (Reserved Word)

Disclaimer

This reserved word removes from the chart the text object with the ID number that matches the one specified. It is important to remember that if an invalid ID number is used, the re served word will return a value of -2 and no additional operations will be performed on any text objects by the strategy, analysis technique, or function that generated the error.

```
Value1 = Text_Delete(Text_ID)
```

`Text_ID` is a numeric expression representing the identification number of the text object to delete.

`Value1` is any numeric variable or array. You must assign the text object reserved word to a numeric variable or array so that you can determine whether or not the reserved word per formed its operation successfully.

Remarks

If the text cannot be deleted, `Text_Delete` returns an error code. When the text is successfully deleted, it returns 0. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Example

The following deletes the text object with the identification number 3:

```
Text_Delete(3);
```

To obtain the error code (or 0 when no error) returned by the reserved word, you can assign the reserved word to a numeric variable, as follows:

```
Value1 = Text_Delete(3) ;
```

Additional Example

The following statements write the text string "Key" wherever there is a key reversal pat tern on the price chart, and delete old text from the chart as new key reversals are found:

```
Variables: OldKeyID(-1), ID(-1);
If Low < Low[1] AND Close > High[1] Then Begin
 OldKeyID = ID;
 ID = Text_New(Date, Time, Low, "Key");
 If OldKeyID <> -1 Then
  Value1 = Text_Delete(OldKeyID);
End;
```

In the above example, we declare two variables to hold the Text IDs for the existing and new text objects. When we find a new key reversal, we assign the ID number of the current text object to the variable `OldKeyID`. We then create a new text object at the new key reversal. Finally, we delete the text object with the ID number held in the variable `OldKeyID`. We first check for `OldKeyID` to be -1, because it will be -1 until we draw the second text object on the chart, and we don’t want to reference a text object that doesn’t exist.

Text_GetActive (Reserved Word)

Disclaimer

This reserved word returns a numeric value representing the ID of the currently active text object.

```
Value1 = Text_GetActive();
```

`Value1` is any numeric variable or array. You must assign the trendline reserved word to a numeric variable or array so that you can determine whether or not the reserved word performed its operation successfully.

Remarks

When the reserved word performs its operation successfully, a 0 is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Text_GetColor (Reserved Word)

Disclaimer

This reserved word returns a numeric expression corresponding to the color assigned to a specified text object. It is important to remember that if an invalid ID number is used, it will return a value of -2 and no additional operations will be performed on any text objects by the strategy, analysis technique, or function that generated the error.

```
Value1 = Text_GetColor(Text_ID)
```

`Text_ID` is a numeric expression representing the ID number of the text object for which to obtain the color.

`Value1` is any numeric variable or array. You must assign the text object reserved word to a numeric variable or array so that you can determine whether or not the reserved word performed its operation successfully.

Remarks

For a list of possible colors and their corresponding numeric values, see EasyLanguage Colors.

When the reserved word performs its operation successfully, it returns the numeric value corresponding to the color. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Example

For example, the following statements use the text string "Key" wherever there is a key reversal pattern on the price chart, and then compare the color of the text object with the back ground of the price chart. If the colors match, the indicator draws the text string using a dif ferent color:

```
   Variables: ID(-1), TxtColor(0);

If Low < Low[1] AND Close > High[1] Then Begin
 ID = Text_New(Date, Time, Low, "Key");
 TxtColor = Text_GetColor(ID);
 If TxtColor = GetBackGroundColor Then
  Value1 = Text_SetColor(ID, TxtColor + 1);
End;
```

In the above example, we first declare two variables, one to hold the text object ID number, the second to hold the number representing the color of the text object. Then, when we find a key reversal, we draw the text object at the low of the bar. We also obtain the color of the text object, and then compare the text object color to the color of the chart background. If it is the same, we change the color of the text object (add one to the existing color number).

Text_GetDate (Reserved Word)

Disclaimer

This reserved word returns a numeric expression corresponding to the EasyLanguage date of the bar on which the specified text object is drawn. It is important to remember that if an invalid ID number is used, it will return a value of -2 and no additional operations will be performed on any text objects by the strategy, analysis technique, or function that generated the error.

```
Value1 = Text_GetDate(Text_ID)
```

`Text_ID` is a numeric expression representing the ID number of the text object whose date you want to obtain.

`Value1` can be any numeric variable or array. The EasyLanguage date obtained is assigned to this variable or array.

Remarks

When the reserved word performs its operation successfully, the date is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Example

To obtain the date returned by the reserved word, you need to assign the reserved word to a numeric variable. The text object with the identification number 1 begins on February 14, 1999. The following returns 990214.

```
Value1 = Text_GetDate(1) ;
```

Additional Example

The following statement assigns to the variable `Value1` the EasyLanguage date of the bar where the text object with the ID number 5 is drawn:

```
   Value1 = Text_GetDate(5);
```

Text_GetFirst (Reserved Word)

Disclaimer

You can draw text objects using trading strategies, analysis techniques (indicators and studies) or functions, or by using the drawing object tool. EasyLanguage enables you to search for text objects based on how they were created.

This reserved word returns the ID number of the oldest text object on the price chart (the first drawn). It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any text objects by the strategy, analysis technique, or function that generated the error.

```
Value1 = Text_GetFirst(Num)
```

`Value1` is any numeric variable or array that holds the ID number of the desired text object. `Num` is a numeric expression representing the origin type of the text object. Refer to EasyLanguage Trendline and Text Object Parameters for a list of possible values.

Remarks

When the reserved word performs its operation successfully, the identification number is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Example

You must assign the reserved word to a numeric variable to obtain the identification number (or error code) returned by the reserved word. The following returns the identification number of the first text object drawn on the chart, regardless of how it was drawn.

```
Value1 = Text_GetFirst(3) ;
```

Additional Example

The following statements delete the oldest text object on a price chart drawn by a trading strategy, analysis technique, or function:

```
Value1 = Text_GetFirst(1);
Value2 = Text_Delete(Value1);
```

Note When the oldest (first) text object is deleted, the next oldest (second) text object becomes the first drawn on the price chart, and so on.

Text_GetHStyle (Reserved Word)

Disclaimer

A text object is always anchored to a specific bar. Because of this, there are three possible ways to horizontally align a text object: to the left of the bar where it is drawn, to the right, or centered. This reserved word returns a numeric value indicating the horizontal alignment of the text object.

```
Value1 = Text_GetHStyle(Text_ID)
```

`Text_ID` is or a numeric expression representing the ID number of the text object whose horizontal alignment value you want to obtain.

`Value1` is any numeric variable or array that holds the horizontal alignment of the desired text object. The reserved word can return one of these three values:

Value	Placement
0	Left
1	Right
2	Centered

Remarks

When the reserved word performs its operation successfully, a horizontal alignment setting 0 through 2 is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Example

The following instructions obtain the horizontal alignment of text object #10 and align it to right of the bar:

```
If Text_GetHStyle(10) <> 1 Then
 Value1 = Text_SetStyle(1);
```

Text_GetNext (Reserved Word)

Disclaimer

This reserved word can be used together with the reserved word Text_GetFirst to traverse all the text objects in a price chart. The charting application stores the chronological order of all text objects added to a chart, and this information is made available via EasyLanguage. This reserved word returns the ID number of the text object on the price chart added immediately after the text object specified.

It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any text objects by the trading strategy, analysis technique, or function that generated the error.

```
Value1 = Text_GetNext(Text_ID, Num)
```

Value1 is any numeric variable or array that holds the ID number of the desired text object. Num is a numeric expression representing the origin type of the text object. Refer to EasyLanguage Trendline and Text Object Parameters for a list of possible values.

Remarks

When the reserved word performs its operation successfully, the identification number for the text is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Example

To obtain the identification number returned by the reserved word, you need to assign the reserved word to a numeric variable. The following returns the identification number of the text drawn on the chart after the text with the identification number 0, and by the text drawing object.

```
Value1 = Text_GetNext(0, 2) ;
```

Additional Example

The following statements set the color of all text objects in the chart to yellow:

```
Value1 = Text_GetFirst(3);
While Value1 <> -2 Begin
 Value2 = Text_SetColor(Value1, Yellow);
 Value1 = Text_GetNext(Value1, 3);
End;
```

In the above example, we obtain the ID number for the first text object drawn on the chart. Then, we set its color to yellow. We then obtain the ID number of the next text object and set that to yellow. This loop continues until Text_GetNext returns -2 indicating that there are no more text objects on the chart. Keep in mind that once the trading strategy, analysis technique, or function returns -2, it cannot draw any more text objects on the chart. In this situation, you may want to use one trading strategy, analysis technique, or function to draw the text objects, and another to change their color.

Text_GetString (Reserved Word)

Disclaimer

This reserved word returns the text string expression corresponding to the text object specified. It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any text objects by the trading strategy, analysis technique, or function that generated the error.

```
MyText = Text_GetString(Text_ID)
```

`Text_ID` is or a numeric expression representing the ID number of the text object whose text string expression you want to obtain.

`MyText` is any text variable or array, and holds the text string expression corresponding to the text object with the ID number specified.

Remarks

When the reserved word performs its operation successfully, the text string is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Example

The following returns the text string used by the text object with the identification number 1. In order to obtain the text string, you must assign the reserved word to a string variable, as follows:

```
Variable: MyText ("") ;
MyText = Text_GetString(1);
```

Note EasyLanguage doesn’t provide pre-declared string variables so you will have to declare your own string variable in order to assign the reserved word to it.

Additional Example

The following statement prints the contents of text object #5 to the Print Log:

```
Print( Text_GetString(5) );
```

Text_GetTime (Reserved Word)

Disclaimer

This reserved word returns a numeric expression corresponding to the EasyLanguage time of the bar on which the specified text object is anchored. It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any text objects by the trading strategy, analysis technique, or function that generated the error.

```
Value1 = Text_GetTime(Text_ID)
```

`Text_ID` is a numeric expression representing the ID number of the text object for which you want to obtain the time.

`Value1` is any numeric variable or array, and holds the time of the specified text object.

Remarks

When the reserved word performs its operation successfully, the time at which the text object is anchored is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Example

The following statement assigns the EasyLanguage time of the bar where the text object with the ID number 5 is drawn to the variable `Value1.` To obtain the time returned by the reserved word, you need to assign the reserved word to a numeric variable.

```
   Value1 = Text_GetTime(5);
```

Additional Example

The following returns 1300 when the text object with the identification number 1 is anchored at 1:00pm.

```
   Value1 =Text_GetTime(1) ;
```

Text_GetValue (Reserved Word)

Disclaimer

Text objects are drawn at a specific price value on the price chart. This reserved word returns a numeric value corresponding to the price at which the specified text object is anchored.

```
Value1 = Text_GetValue(Text_ID)
```

`Text_ID` is a numeric expression representing the ID number of the text object whose price value you want to obtain.

`Value1` is any numeric variable or array, and holds the price value at which the specified object is anchored.

Remarks

When the reserved word performs its operation successfully, the price value at which the text object is anchored is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Examples

The following returns a value of 41.543 when the text object with the identification number 5 is anchored at a price value of 41.543. To obtain the price value returned by the reserved word, you need to assign the reserved word to a numeric variable.

```
Value1 = Text_GetValue(5) ;
```

The following statement can be used to print to the Print Log the value at which text object 10 is drawn:

```
Print( Text_GetValue(10) ) ;
```

Text_GetVStyle (Reserved Word)

Disclaimer

A text object is always anchored at a specific price value on a price chart, and there are three possible ways to align the text object vertically: the top being at the specified price, the bottom being at the specified price, or centered. This reserved word returns a numeric value representing the vertical alignment of the specified text object.

```
Value1 = Text_GetVStyle(Text_ID)
```

`Text_ID` is a numeric expression representing the ID number of the text object whose vertical alignment you want to obtain.

`Value1` can be any numeric variable or array, and holds the price value at which the specified object is anchored.

This reserved word returns one of three values:

Value	Placement
0	Top
1	Bottom
2	Centered

Remarks

When the reserved word performs its operation successfully, a vertical alignment setting 0 through 2 is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Example

To obtain the vertical alignment setting returned by the reserved word, you need to assign the reserved word to a numeric variable. When the text object with the identification number 1 is centered, the following returns a value of 2.

```
Value1 = Text_GetVStyle(1) ;
```

Additional Example

The following instructions obtain the vertical alignment of text object #10 and set it to the bottom:

```
If Text_GetHStyle(10) <> 1 Then
 Value1 = Text_SetStyle(1);
```

Text_New (Reserved Word)

Disclaimer

This reserved word adds the specified text string to a price chart, at the specified bar and price value. It returns a numeric expression corresponding to the ID number of the text object added to the chart. If you want to modify the text object in any way, it is very important that you capture and keep this number; the ID number is the only way of referencing a specific text object.

```
Value1 = Text_New(BarDate, BarTime, Price, "MyText")
```

`BarDate` and `BarTime` are numeric expression corresponding to the date and time, respectively, for the bar on which you want to anchor the text object. `Price` is a numeric expression representing the price value at which to anchor the text object and `MyText` is the text string expression to add to the price chart.

All text objects are anchored at a specific bar and price value on the price chart. You need to provide this information to the `Text_New` reserved word in order for the trading strategy, analysis technique, or function to add a text object to the chart.

`Value1` is any numeric variable or array, and holds the ID number for the new text object.

Text objects are added to the chart using the default color, and vertical and horizontal alignment of the charting application. Any of these properties can be changed using other Text-related reserved words.

Remarks

When the reserved word performs its operation successfully, the reserved word returns the ID number for the new text object. When the reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note When any drawing object error occurs, no additional operations will be performed on any drawing objects by the trading strategy, analysis technique, or function that generated the error.

Example

For example, the following statements add a text string "Key" to a price chart every time there is a key reversal pattern*:*

```
Variable: ID(-1);
If Low < Low[1] AND Close > Close[1] Then
   ID = Text_New(Date, Time, Low, "Key");
```

Text_SetColor (Reserved Word)

Disclaimer

This reserved word sets the color of the specified text object.

```
Value1 = Text_SetColor(Text_ID, Color)
```

`Text_ID` is a numeric expression representing the ID number of the text object, and `Color` is an EasyLanguage color or its numeric equivalent. See EasyLanguage Colors for a list of valid colors.

`Value1` is any numeric variable or array. You must assign the text object reserved word to a numeric variable or array so that you can determine whether or not the reserved word performed its operation successfully.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any text objects by the trading strategy, analysis technique, or function that generated the error. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Example

To set the text object with the identification number 1 to yellow, you would write either of the following:

```
Text_SetColor(1, 7);
```

```
Text_SetColor(1, Yellow);
```

Additional Example

The following indicator displays the word "Key" wherever there is a key reversal pattern on the price chart, and compares the color of the text object with the background of the price chart. If the colors match, the indicator sets the text object to a different color (it adds 1 to the current color):

```
Variables: ID(-1), TxtColor(0);

If Low < Low[1] AND Close > High[1] Then Begin
 ID = Text_New(Date, Time, Low, "Key");
 TxtColor = Text_GetColor(ID);
 If TxtColor = GetBackgroundColor Then
  Value1 = Text_SetColor(ID, TxtColor + 1);
End;
```

Text_SetLocation (Reserved Word)

Disclaimer

All text objects are anchored at a specific bar and price value on the price chart. This reserved word modifies the point at which the specified text object is anchored.

```
Value1 = Text_SetLocation(Text_ID, BarDate, BarTime, Price)
```

`Text_ID` is a numeric expression representing the ID number of the text object to modify; `BarDate` and `BarTime` are numeric expressions representing the new EasyLanguage date and time, respectively, at which to anchor the text object; and `Price` is the new price value at which to anchor the text object.

`Value1` is any numeric variable or array. You must assign the text object reserved word to a numeric variable or array so that you can determine whether or not the reserved word performed its operation successfully.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any text objects by the trading strategy, analysis technique, or function that generated the error. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

We recommend that you change the location of the text object rather than delete the text object and draw a new one. Relocating an existing object is faster and generates fewer ID numbers to keep track of.

Example

The following moves the text object with the identification number 2 to January 14, 1999, 3:00pm, at a price location of 150.

```
Text_SetLocation(2, 990114, 1500, 150.00) ;
```

Additional Example

The first statement is done only once, on the first bar after MaxBarsBack. This statement creates a new text object containing the symbol and places it above the first

bar after Max BarsBack. The second statement is done on every bar (or tick) and changes the location of the text so that it always displays above the last visible bar of the chart:

```
If BarNumber = 1 Then Value1 = Text_New(Date, Time, High + Range*.25,
GetSymbolName);
Value2 = Text_SetLocation(Value1, Date, Time, High + Range*.25);
```

Text_SetString (Reserved Word)

Disclaimer

This reserved word changes the text string expression of the specified text object.

```
Value1 = Text_SetString(Text_ID, "MyText")
```

`Text_ID` is a numeric expression representing the ID number of the text object whose text string expression you want to modify, and `MyText` is the new text string expression for the text object.

`Value1` is any numeric variable or array. You must assign the text object reserved word to a numeric variable or array so that you can determine whether or not the reserved word performed its operation successfully.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any text objects by the trading strategy, analysis technique, or function that generated the error. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

We recommend that you change the text string expression of the text object rather than delete the text object and draw a new one. Changing an existing text object is faster and generates fewer ID numbers to keep track of.

Example

The following is used to change the string in the text object with the identification number 1 to `New String`.

```
Text_SetString(1, "New String") ;
```

Additional Example

These statements display the closing price of the symbol above the first bar in the chart (af ter `MaxBarsBack`) and then change the location of the text and the text to always display the closing price of the last bar on the chart:

```
If BarNumber = 1 Then
 Value1 = Text_New(Date, Time, High * 1.01, NumToStr(Close, 2) );

Value2 = Text_SetLocation(Value1, Date, Time, High * 1.01);
Value3 = Text_SetString(Value1, NumToStr(Close, 2) );
```

Text_SetStyle (Reserved Word)

Disclaimer

A text object is always anchored at a specific bar and price value. There are three horizontal placement settings for the text object: to the left of the text object's bar, to the right, or centered. Also, there are three vertical placement settings: above the specified price, below the specified price, or centered on the specified price.

This reserved word sets the horizontal and vertical placement of the specified text object.

```
Value1 = Text_SetStyle(Text_ID, Horiz, Vert);
```

`Text_ID` is a numeric expression representing the ID number of the text object whose placement you want to change, and `Horiz` and `Vert` are numeric expressions representing the horizontal and vertical placement of the text object, respectively.

`Value1` is any numeric variable or array. You may assign the text object reserved word to a numeric variable or array so that you can determine whether or not the reserved word performed its operation successfully.

You can use one of three horizontal placement values (`Horiz`):

Value	Placement
0	To the right of the bar specified for the text object
1	To the left of the bar specified for the text object
2	Centered on the bar specified for the text object

You can use one of three vertical placement values (`Vert`):

Value	Placement
0	Beneath the price specified for the text object
1	Above the price specified for the text object
2	Centered on the specified price location of the text object

If there are no text objects with the ID number you specify, or if the operation fails in any way, this reserved word will return a numeric expression corresponding to one of the EasyLanguage drawing objects error codes, and no additional operations will be performed on any text objects by the trading strategy, analysis technique, or function that generated the error. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Example

The following statement changes the placement of text object #3 so it is drawn to the left of the specified bar (based on date/time) and sits above the specified price:

```
Text_SetStyle(3, 1, 1) ;
```

Than (Reserved Word)

Disclaimer

THAN is a Skip Word that performs no function within your code. It is used within a statement only to make the EasyLanguage code more readable.

Remarks

Skip words in EasyLanguage are not necessary, however, they sometimes make it easier to understand the purpose of the code.

Syntax Coloring in the EasyLanguage PowerEditor is dark green for all skip words by default.

Example

```
If the Close > than 100 Then
      Alert;
```

The word "than" is not necessary and the code functions the same way regardless of its presence.

The (Reserved Word)

Disclaimer

THE is a Skip Word that performs no function within your code. It is used within a statement only to make the EasyLanguage code more readable.

Remarks

Skip words in EasyLanguage are not necessary, however, they sometimes make it easier to understand the purpose of the code.

Syntax Coloring in the EasyLanguage PowerEditor is dark green for all skip words by default.

Example

```
If the Close > than the High[1] Then
      Alert;
```

The word "the" is not necessary and the code functions the same way regardless of its presence.

Then (Reserved Word)

Disclaimer

This word is used to introduce a condition that will be evaluated to determine execution of additional code.

```
Then
```

Remarks

The second part of an `If… Then` or `If… Then… Else` statement. `If… Then` statements allow you to check a true/false condition, and then take a specific action depending on the outcome of the condition. If the condition is true, the action after "Then" is executed.

Examples

```
If Condition1 Then
 (Your Code Line1}
```

`Then` is used here to designate the Line1 as the code that will be executed based on the outcome of `Condition1`. The Line1 code will be executed if `Condition1` returns a value of `True`. If `Condition1` is false, the Line1 code will not be executed.

```
If Condition1 And Condition2 Then Begin
 {Your Code Line1}
 {Your Code Line2, etc.}
End
Else Begin
 {Your Code Line3}
 {Your Code Line4, etc.}
End;
```

`Then` is used here to designate the Line1 as the code that will be executed based on the outcome of `Condition1`. The Line1 and Line2 code will be executed if `Condition1` and `Condition2` return a value of `True`. If `Condition1` or `Condition2` is false, the Line3 and Line4 code will be executed.

THEORETICALGROSSIN (Reserved Word)

Disclaimer

THEORETICALGROSSIN is used in OptionStation to return the theoretical gross cost for entering a position.

THEORETICALGROSSOUT (Reserved Word)

Disclaimer

THEORETICALGROSSOUT is used in OptionStation to return the theoretical gross proceeds for exiting a position.

THEORETICALVALUE (Reserved Word)

Disclaimer

THEORETICALVALUE is used in OptionStation to return the modeled theoretical value of an option.

THETA (Reserved Word)

Disclaimer

VEGA is used in OptionStation to return the risk value Theta of an option or position.

This (Reserved Word)

Disclaimer

Reserved word used to refer to the present bar.

```
{Action} This Bar on Close
```

Remarks

This is normally used with `Bar` or `Day` to specify the current Bar or Day.

Examples

```
  Buy This Bar on Close
```

refers to a Long Entry placed on the Close of the current bar

```
  Sell This Bar on Close
```

refers to a Short Entry placed on the Close of the current bar

Thursday (Reserved Word)

Disclaimer

Returns the number 4 as the value for Thursday.

```
Thursday
```

Example

```
   Thursday
```

…returns a value of 4 as the value for Thursday.

Ticks (Reserved Word)

Disclaimer

Reserved word that represents the combined number of `Upticks` and `Downticks`.

Remarks

`Ticks` can be used in place of `Upticks + Downticks`. See EasyLanguage Reserved Words Related to Ticks, Volume & Open Interest for more information.

TICKTYPE (Reserved Word)

Disclaimer

TICKTYPE is an OptionStation reserve word that returns the type of price update that triggered the current core calculation event.

Time (Reserved Word)

Disclaimer

This reserved word returns a numeric expression representing the EasyLanguage time (HHMM format) of the closing price of the current bar.

Remarks

`Time` returns the time in 24-hour format.

Examples

`Time` returns 1600 if the Time of the bar is 4:00pm.

`Time` returns 0930 if the Time of the bar is 9:30am.

Additional Example

For example, you can write your strategy, analysis technique, or function such that it only evaluates the EasyLanguage instructions when the trade time is less than 11:00am:

```
If Time < 1100 Then
 { EasyLanguage instruction } ;
```

TimeToString (Reserved_Word)

Disclaimer

This reserved word returns a string expression representing only the time portion of the specified DateTime value.

```
TimeToString(dDateTime);
```

Where `dDateTime` is a double-precision decimal expression representing the Julian Date (since 1900) and time (since midnight).

Remarks

If a specific time is entered for the numeric expression, it must be expressed in DateTime format.

Examples

`strVar = DateToString(37561.61200);` returns a value of "2:41:17 PM" for the specified DateTime of 37561.61200.

TL_Delete (Reserved Word)

Disclaimer

This reserved word deletes the specified trendline from the price chart.

```
Value1 = TL_Delete(Tl_ID)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline to delete.

`Value1` is any numeric variable or array.

Note When the trendline is successfully deleted, it returns 0. It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Example

The following deletes the trendline with the identification number 3:

```
TL_Delete(3);
```

To obtain the error code (or 0 when no error) returned by the reserved word, you can assign the reserved word to a numeric variable, as follows:

```
Value1 = TL_Delete(3) ;
```

Additional Example

The following statements draw a trendline at the low of a key reversal and extend it to the right, and in addition, delete the old trendline from the chart when a new key reversal is found:

```
Variables: OldKeyID(-1), ID(-1);

If Low < Low[1] AND Close > High[1] Then Begin
 OldKeyID = ID;
 ID = TL_New(Date[1], Time[1], Low, Date, Time, Low);
 Value1 = TL_SetExtRight(ID, True);
 If OldKeyID <> -1 Then
  Value1 = TL_Delete(OldKeyID);
End;
```

In the above example, first we declare two variables, one to hold the ID number of the old trendline, and one to hold the ID number for the new trendline. When we find a new key reversal, we store the existing trendline's ID number in `OldKeyID`, and create a new trendline at the low of the key reversal bar and extend it to the right. Then, we delete the old trendline. Before deleting the old trendline, we first check to make sure the ID number in `OldKeyID` is not -1, which it will be until the second trendline is drawn. This way, we don't reference an invalid ID number.

TL_GetActive (Reserved Word)

Disclaimer

This reserved word returns a numeric value representing the ID of the currently active trendline..

```
  Value1 = TL_GetActive();
```

`Value1` is any numeric variable or array. You must assign the trendline reserved word to a numeric variable or array so that you can determine whether or not the reserved word performed its operation successfully.

Remarks

When the reserved word performs its operation successfully, a 0 is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

TL_GetAlert (Reserved Word)

Disclaimer

This reserved word obtains the alert setting for the specified trendline.

```
   Value1 = TL_GetAlert(Tl_ID)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline whose alert status you want to obtain.

`Value1` can be any numeric variable or array, and holds the alert status. This reserved word returns one of these three values:

Value	Description
0	None – no alert enabled
1	Breakout Intrabar
2	Breakout on Close

Remarks

When the reserved word performs its operation successfully, it returns a value of 0, 1 or 2, indicating the alert status. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note An alert set to *Breakout on Close* is triggered when on a previous bar, the close of the symbol lies on one side of the trendline, and on the current bar, the close ends up on the other side of the trendline. This type of alert is only evaluated once the bar is closed.

An alert set to *Breakout Intrabar* is triggered if the high crosses over the trendline or if the low crosses under the trendline. This alert is triggered at the moment the trendline is broken.

Example

To obtain the alert status returned by the reserved word, you need to assign the reserved word to a numeric variable. The following obtains the alert status for the trendline referenced by `Value99`.

```
   Value1 = TL_GetAlert(Value99);
```

Additional Example

The following statement checks the alert status for the trendline referenced by `Value99` and if it is not set to *Breakout on Close*, it enables it and sets it to *Breakout on Close*:

```
   If TL_GetAlert(Value99) <> 2 Then
    Value1 = TL_SetAlert(Value99, 2);
```

TL_GetBeginDate (Reserved Word)

Disclaimer

This reserved word returns the date of the starting point of the trendline. The start point is the one with the earlier date. If the trendline is vertical, the lower of the two points is considered to be the starting point.

```
Value1 = TL_GetBeginDate(Tl_ID)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline whose start date you want to obtain.

`Value1` is any numeric variable or array, and holds the date of the starting point. The date is returned using YYMMDD or YYYMMDD format (three digits are used to express the year 2000 and higher).

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Example

The trendline referenced by `Value99` begins on January 14, 1999. The following will return 990114:

```
Value1 = TL_GetBeginDate(Value99) ;
```

TL_GetBeginTime (Reserved Word)

Disclaimer

This reserved word returns the time of the starting point of the trendline. The start point is the one with the earlier date. If the trendline is vertical, the lower of the two points is considered to be the starting point.

```
Value1 = TL_GetBeginTime(Tl_ID)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline whose starting time you want to obtain.

`Value1` is any numeric variable or array, and holds the time of the starting point.

Remarks

When the reserved word performs its operation successfully, it returns the time. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

To obtain the time returned by the reserved word, you need to assign the reserved word to a numeric variable. The starting point for the trendline referenced by `Value99` is at 1:00pm. The following will return 1300.

```
Value1 = TL_GetBeginTime(Value99);
```

TL_GetBeginVal (Reserved Word)

Disclaimer

This reserved word returns a numeric expression corresponding to the price value used as the starting point of the trendline. The starting point of the trendline is the one with the earlier date; if the trendline is vertical, the lower of the two points is considered to be the starting point.

```
Value1 = TL_GetBeginVal(Tl_ID)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline whose starting price value you want to obtain.

`Value1` is any numeric variable or array, and holds the price value of the starting point of the trendline.

Remarks

When the reserved word performs its operation successfully, it returns the price value. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

To obtain the price value returned by the reserved word, you need to assign the reserved word to a numeric variable. The trendline referenced by `Value99` begins at a price of 41.543. The following will return 41.543.

```
Value1 = TL_GetBeginVal(Value99);
```

TL_GetColor (Reserved Word)

Disclaimer

This reserved word returns a numeric expression corresponding to the color assigned to the specified trendline.

```
Value1 = TL_GetColor(Tl_ID)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline whose color you want to obtain.

`Value1` is any numeric variable or array, and holds the EasyLanguage color or numeric equivalent of the specified trendline.

Remarks

For a list of possible colors and their corresponding numeric values, see EasyLanguage Colors.

When the reserved word performs its operation successfully, it returns the numeric value corresponding to the color. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

To obtain the numeric color value returned by the reserved word, you need to assign the reserved word to a numeric variable. The trendline referenced by `Value99` is yellow. Therefore, the following returns the value 7.

```
Value1 = TL_GetColor(Value99);
```

Additional Example

The following statements draw a trendline at the low of each key reversal pattern. If the col or of the trendline matches the background color of the chart, the indicator sets the trendline to a different color (it adds 1 to the current color):

```
Variable: ID(-1);

If Low < Low[1] AND Close > High[1] Then Begin
 ID = TL_New(Date[1], Time[1], Low, Date, Time, Low);
 Value1 = TL_GetColor(ID);
 If Value1 = GetBackGroundColor Then
  Value2 = TL_SetColor(ID, Value1 + 1);
End;
```

TL_GetEndDate (Reserved Word)

Disclaimer

This reserved word returns the date of the ending point of the trendline. The ending point of the trendline is the one with the later date; if the trendline is vertical, the higher of the two points is considered to be the ending point.

```
Value1 = TL_GetEndDate(Tl_ID)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline whose end date you want to obtain.

`Value1` is any numeric variable or array, and holds the date of the starting point.

Remarks

When the reserved word performs its operation successfully, the date is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

The following statement assigns the EasyLanguage date of the bar used as the end point for the trendline referenced by `Value99` to the variable `Value1`:

```
Value1 = TL_GetEndDate(Value99);
```

TL_GetEndTime (Reserved Word)

Disclaimer

This reserved word returns the time of the ending point of the trendline. The ending point of the trendline is the one with the later time; if the trendline is vertical, the higher of the two points is considered to be the ending point.

```
Value1 = TL_ GetEndTime(Tl_ID)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline whose ending time you want to obtain.

`Value1` is any numeric variable or array, and holds the time of the starting point.

Remarks

When the reserved word performs its operation successfully, the time is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

The following statement assigns the EasyLanguage time of the bar used as the end point for the trendline referenced by `Value99` to the variable `Value1`:

```
Value1 = TL_GetEndTime(Value99);
```

TL_GetEndVal (Reserved Word)

Disclaimer

This reserved word returns a numeric expression corresponding to the price value used as the ending point of the trendline. The ending point of the trendline is the one with the later date; if the trendline is vertical, the higher of the two points is considered to be the ending point.

```
Value1 = TL_GetEndVal(Tl_ID)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline whose ending price value you want to obtain.

`Value1` is any numeric variable or array, and holds the price value of the ending point of the trendline.

Remarks

When the reserved word performs its operation successfully, the price value is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

To obtain the price value returned by the reserved word, you need to assign the reserved word to a numeric variable. The trendline referenced by `Value99` ends at a price of 41.543. The following returns a value of 41.543.

```
Value1 = TL_GetEndVal(Value99);
```

TL_GetExtLeft (Reserved Word)

Disclaimer

Trendlines can be extended to the right or left. This reserved word returns a value of True or False. If the trendline is extended to the left, it will return a value of True; otherwise, it will return a value of False.

```
   Condition1 = TL_GetExtLeft(Tl_ID)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline whose extension status you want to obtain.

`Condition1` can be any true/false variable or array, and holds the true/false value determin ing whether or not the trendline is extended. If an invalid ID number is used, the value `False` is returned.

Remarks

When the reserved word performs its operation successfully, the true or false value is returned, indicating whether or not the trendline is extended left. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Example

To obtain the true/false value returned by the reserved word, you need to assign the reserved word to a true/false variable. The trendline referenced by `Value99` is not extended to the left. The following returns the value false.

```
Condition1 = TL_GetExtLeft(Value99);
```

Additional Example

The following instructions extend the trendline referenced by `Value99` to the left if it is not already extended:

```
   If TL_GetExtLeft(Value99) = False Then
    Value1 = TL_SetExtLeft(Value99, True);
```

TL_GetExtRight (Reserved Word)

Disclaimer

Trendlines can be extended to the right or left. This reserved word returns a value of True or False. If the trendline is extended to the right, it will return a value of True; otherwise, it will return a value of False.

```
Condition1 = TL_GetExtRight(Tl_ID);
```

`Tl_ID` is a numeric expression representing the ID number of the trendline whose extension status you want to obtain.

`Condition1` can be any true/false variable or array, and holds the true/false value determin ing whether or not the trendline is extended. If an invalid ID number is used, the value `False` is returned.

Remarks

When the reserved word performs its operation successfully, the true or false value is returned, indicating whether or not the trendline is extended right. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Example

To obtain the true/false value returned by the reserved word, you need to assign the reserved word to a true/false variable. The trendline referenced by `Value99` is extended to the right. The following returns the value true.

```
Condition1 = TL_GetExtRight(Value99);
```

Additional Example

The following instructions extend the trendline referenced `Value99`by to the right if it is not already extended:

```
If TL_GetExtRight(Value99) = False Then
 Value1 = TL_SetExtRight(Value99, True);
```

TL_GetFirst (Reserved Word)

Disclaimer

EasyLanguage enables you to search for trendlines based on how and in what order they were created. The charting application stores the chronological order of all trendlines added to a chart, and this information is made available to EasyLanguage. This reserved word returns the ID number of the first trendline added to the price chart (by a trading strategy, analysis technique, or function, or by a drawing tool, or by either).

```
Value1 = TL_GetFirst(Pref)
```

`Value1` is any numeric variable or array that holds the ID number of the desired trendline. `Pref` is a numeric expression representing the origin type of the trendline. Refer to EasyLanguage Trendline and Text Object Parameters for a list of possible values.

Remarks

When the reserved word performs its operation successfully, the identification number is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note When the oldest (first) trendline is deleted, the next oldest (second) trendline becomes the first drawn on the price chart, and so on.

It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

You must assign the reserved word to a numeric variable to obtain the identification number (or error code) returned by the reserved word. The following returns the identification number of the first trendline drawn on the chart, regardless of how it was drawn.

```
Value1 = TL_GetFirst(3) ;
```

Additional Example

The following statements delete the oldest trendline on a price chart drawn by a trading strategy, analysis technique, or function:

```
Value1 = TL_GetFirst(1);
Value2 = TL_Delete(Value1);
```

TL_GetNext (Reserved Word)

Disclaimer

EasyLanguage enables you to search for trendlines based on how they were created. The charting application stores the chronological order of all trendlines added to a chart, and this information is made available to EasyLanguage. This reserved word returns the ID number of the trendline on the price chart added immediately after the trendline specified. You can use this reserved word together with the reserved word `TL_GetFirst` to traverse all the trendlines in a price chart.

```
Value1 = TL_GetNext(TL_ID, Num)
```

`Value1` is any numeric variable or array that holds the ID number of the desired trendline. `Num` is a numeric expression representing the origin type of the trendline. Refer to EasyLanguage Trendline and Text Object Parameters for a list of possible values.

Remarks

When the reserved word performs its operation successfully, the identification number for the trendline is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

To obtain the identification number returned by the reserved word, you need to assign the reserved word to a numeric variable. The following returns the identification number of the trendline drawn on the chart after the trendline referenced by `Value99`, and by the trendline drawing object.

```
Value1 = TL_GetNext(Value99, 2) ;
```

Additional Example

The following statements set the color of all trendlines in the chart to yellow:

```
Value1 = TL_GetFirst(3);
While Value1 <> -2 Begin
 Value2 = TL_SetColor(Value1, Yellow);
 Value1 = TL_GetNext(Value1, 3);
End;
```

In the above example, we obtain the ID number for the first trendline drawn on the chart. Then, we set its color to yellow. We then obtain the ID number of the next trendline and set that to yellow. This loop continues until *TL_GetNext* returns -2 indicating that there are no more trendlines on the chart. Keep in mind that once the trading strategy, analysis technique, or function returns -2, it cannot draw any more trendline on the chart. In this situation, you may want to use one trading strategy, analysis technique, or function to draw the trendlines, and another to change their color.

TL_GetSize (Reserved Word)

Disclaimer

This reserved word returns a numeric expression representing the thickness of the trendline, where 0 is the thinnest, and 6 is the thickest.

```
   Value1 = TL_GetSize(Tl_ID)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline whose thickness setting you want to obtain.

`Value1` can be any numeric variable or array, and holds the thickness setting.

Remarks

Line thickness ranges from 0 (thinnest) to 6 (thickest). These correspond to the seven selections in the **Weight** drop-down list in the **Style** tab of the **Format Trendline** dialog box.

When the reserved word performs its operation successfully, a line thickness setting 0 through 6 is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

To obtain the line thickness value returned by the reserved word, you need to assign the reserved word to a numeric variable. When the trendline referenced by `Value99` is drawn with the third heaviest weight. The following returns a value of 2.

```
   Value1 = TL_GetSize(Value99);
```

TL_GetStyle (Reserved Word)

Disclaimer

This reserved word returns a numeric expression representing the line style used for the specified trendline.

```
Value1 = TL_GetStyle(Tl_ID)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline whose line style you want to obtain.

`Value1` is any numeric variable or array, and holds the numeric expression representing the line style of the specified trendline. Following are the possible return values and their numeric equivalents:

Style Value	Style Reserved Word
1	Tool_Solid
2	Tool_Dashed
3	Tool_Dotted
4	Tool_Dashed2
5	Tool_Dashed3

These correspond to the five selections in the **Style** drop-down list in the **Style** tab of the **Format Trendline** dialog box. You can use either the numbers or the EasyLanguage reserved word.

Remarks

When the reserved word performs its operation successfully, it returns the number corresponding to the line style setting. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

To obtain the line style value returned by the reserved word, you need to assign the reserved word to a numeric variable. The trendline referenced by `Value99` is drawn using a solid line style. The following returns 1.

```
Value1 = TL_GetStyle(Value99);
```

Additional Example

The following `If-Then` statement verifies that a trendline is solid before executing the EasyLanguage instruction:

```
If TL_GetStyle(Value99) = Tool_Solid Then
 {EasyLanguage instruction } ;
```

TL_GetValue (Reserved Word)

Disclaimer

This reserved word returns a numeric expression corresponding to the value of a trendline at a specific bar. It is important to remember that this reserved word returns a value even if the trendline is not shown on or projected through the bar specified. For example, if a trendline is drawn from December 1st to January 5th, and the following statement is used:

```
Value1 = TL_GetValue(10, 990203, 1400);
```

Even though the date specified is in February, the `TL_GetValue` reserved word will return the trendline value as if the trendline were extended to that particular bar (along the same slope).

```
Value1 = TL_GetValue(Tl_ID, TLDate, TLTime)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline whose price value you want to obtain. `TLDate` and `TLTime` are the date and time, respectively, of the bar for which you want to obtain the trendline's value.

`Value1` is any numeric variable or array. You must assign the trendline reserved word to a numeric variable or array so that you can determine whether or not the reserved word performed its operation successfully.

Remarks

When the reserved word performs its operation successfully, the vertical axis (price) value is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

To obtain the price value returned by the reserved word, you need to assign the reserved word to a numeric variable. On January 7, 1999 at 4:00pm, the trendline referenced by `Value99` intersects the price value of 53.350. The following returns 53.350.

```
Value1 = TL_GetValue(Value99, 990107, 1600);
```

Additional Example

The following statement triggers an alert when the close crosses over the trendline referenced by `Value99`:

```
If Close Crosses Over TL_GetValue(Value99, Date, Time) Then
 Alert("Trendline is broken");
```

TL_New (Reserved Word)

Disclaimer

This reserved word adds a trendline with the specified starting and ending points to a price chart. It returns a numeric expression corresponding to the ID number of the trendline added to the chart.

```
Value1 = TL_New(iBarDate, iBarTime, iPrice, eBarDate, eBarTime, ePrice)
```

iBarDate, iBarTime, and iPrice are numeric expressions corresponding to the date, time, and price, respectively, of the starting point; eBarDate, eBarTime, and ePrice are numeric expressions corresponding to the date, time, and price, respectively, of the end point of the trendline. Dates use YYMMDD or YYYMMDD format (three digits are used to express the year 2000 and later), and times use 24-hour format, HHMM.

Value1 is any numeric variable or array, and holds the ID number for the new trendline.

Remarks

When the reserved word performs its operation successfully, the reserved word returns the ID number for the new trendline. When the reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note When any drawing object error occurs, no additional operations will be performed on any drawing objects by the trading strategy, analysis technique, or function that generated the error.

Be aware that a minimum of two different points are needed in order to draw any trendline on a price chart, and this is the information that you need to provide to the TL_New reserved word to draw a trendline on the price chart from a trading strategy, analysis technique, or function.

Trendlines are added to the chart using the default properties set in the charting application. As you will see, you can change any of these properties using other trendline related reserved words. If you want to modify a trendline in any way, it is very important that you capture and keep the number; the ID number is the only way of referencing a specific trendline.

Example

The following creates a trendline that begins a 9:30am on January 7, 1999 at a value of 45, and ends at 4:00pm on January 25, 1999 at a value of 37.250:

```
Value1 = TL_New(990107, 0930, 45, 990125, 1600, 37.250) ;
```

Additional Example

The following statements add a trendline to the price chart (and extend it to the right) every time there is a key reversal pattern*:*

```
Variable: ID(-1);
If Time = SessionFirstBarTime(1, 1) Then
   ID = TL_New(Date, Time, Open, Date, SessionEndTime(1, 1), Open);
```

TL_SetAlert (Reserved Word)

Disclaimer

This reserved word changes the alert status for a trendline.

```
Value1 = TL_SetAlert(Tl_ID, AlertVal)
```

`Tl_ID` is a numeric expression representing the identification number of the trendline, and `AlertVal` is a numeric expression representing the alert setting for the trendline. You can specify one of these three values:

Value	Description
0	None – no alert enabled
1	Breakout Intrabar
2	Breakout on Close
3	Breakout intra/inter-bar

`Value1` is any numeric variable or array. You must assign the trendline reserved word to a numeric variable or array so that you can determine whether or not the reserved word performed its operation successfully.

Remarks

When the reserved word performs its operation successfully, a 0 is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note An alert set to *Breakout on Close* is triggered when on the previous bar, the close of the symbol was lower than the trendline, and on the current bar, the close is higher than the trendline. The alert is also triggered if the reverse scenario occurs- if the close was higher than the trendline, and on the current bar, the close is lower than the trendline. This type of alert is only evaluated once the bar is closed.

An alert set to *Breakout Intrabar* is triggered if the high crosses over the trendline or if the low crosses under the trendline. This alert is triggered at the moment the trendline is broken.

It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

The following sets the alert of the trendline referenced by `Value99` to Breakout Intrabar:

```
Value1 = TL_SetAlert(Value99, 1);
```

Additional Example

The following statement checks the alert status for the trendline referenced by `Value99` and if it is not set to *Breakout on Close*, it enables it and sets it to *Breakout on Close*:

```
If TL_GetAlert(Value99) <> 2 Then
 Value1 = TL_SetAlert(Value99, 2);
```

TL_SetBegin (Reserved Word)

Disclaimer

This reserved word changes the start point of the specified trendline; the start point has an earlier date and time. If the trendline is vertical, the point with the lower price value is considered the starting point.

If the starting point of a trendline is changed (by EasyLanguage or by using the drawing tool) such that it has a later date than the ending point, the starting point then becomes the old ending point of the trendline.

```
Value1 = TL_SetBegin(Tl_ID, iDate, iTime, iVal)
```

`Tl_ID` is a numeric expression representing the identification number of the trendline, and `iDate`, `iTime`, and `iVal` are numeric expressions representing the trendline's starting point date, time, and value respectively. Dates and times use YYMMDD / YYYMMDD and 24-hour format, HHMM, respectively.

`Value1` is any numeric variable or array. You must assign the trendline reserved word to a numeric variable or array so that you can determine whether or not the reserved word performed its operation successfully.

Remarks

When the reserved word performs its operation successfully, a 0 is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Example

The following sets the trendline referenced by `Value99` to a value of 107.225 on February 21, 1999 at 10am.

```
Value1 = TL_SetBegin(Value99, 990221, 1015, 107.225);
```

Additional Example

The following statement sets the start point of the trendline referenced by `Value99` to the high price 10 bars ago:

```
Value1 = TL_SetBegin(Value99, Date[10], Time[10], High[10]);
```

TL_SetColor (Reserved Word)

Disclaimer

This reserved word changes the color of the specified trendline.

```
Value1 = TL_SetColor(Tl_ID, Color)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline whose color you want to change, and `Color` is one of the EasyLanguage supported colors. See EasyLanguage Colors for a list of valid colors.

`Value1` is any numeric variable or array. You must assign the trendline reserved word to a numeric variable or array so that you can determine whether or not the reserved word performed its operation successfully.

Remarks

When the reserved word performs its operation successfully, a 0 is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

To set the color of the trendline referenced by `Value99` to yellow, you would write either of the following:

```
TL_SetColor(Value99, 7);
```

```
TL_SetColor(Value99, Yellow);
```

To obtain the error code (or 0 when no error) returned by the reserved word, you need to assign the reserved word to a numeric variable, as follows:

```
Value1 = TL_SetColor(Value99, 7);
```

Additional Example

The following statements draw a trendline at the low of a key reversal, and compare the color of the trendline with the background of the chart. If the colors match, the EasyLanguage instructions add 1 to the color, and set the trendline to this new color:

```
Variables: ID(-1), TLColor(0);

If Low < Low[1] AND Close > High[1] Then Begin
 ID = TL_New(Date[1], Time[1], Low, Date, Time, Low);
 TLColor = TL_GetColor(ID);
 If TLColor = GetBackgroundColor Then
  Value1 = TL_SetColor(ID, TxtColor + 1);
End;
```

TL_SetEnd (Reserved Word)

Disclaimer

This reserved word changes the end point of the specified trendline; the end point has a later date and time. If the trendline is vertical, the point with the higher price value is considered the ending point.

If the ending point of a trendline is changed (by EasyLanguage or by using the drawing tool) such that it has an earlier date than the starting point, the ending point then becomes the original starting point of the trendline.

```
Value1 = TL_SetEnd(Tl_ID, eDate, eTime, eVal)
```

`Tl_ID` is a numeric expression representing the identification number of the trendline, and `eDate`, `eTime`, and `eVal` are numeric expressions representing the trendline's new ending point date, time, and price value, respectively. Dates and times use YYMMDD / YYYMMDD and 24-hour format, HHMM, respectively.

`Value1` is any numeric variable or array. You must assign the trendline reserved word to a numeric variable or array so that you can determine whether or not the reserved word performed its operation successfully.

Remarks

When the reserved word performs its operation successfully, a 0 is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note This reserved word returns zero (0) when it successfully changes the end point of a trendline, and one of the EasyLanguage drawing object errors when it fails. For example, if the end point of the trendline is set to exactly the same value of the start point, the reserved word will return an error -5. Also, it is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

The following sets the end point of the trendline referenced by `Value99` to a value of 207.125 on February 21, 1999 at 3:15pm.

```
Value1 = TL_SetEnd(Value99, 990221, 1515, 207.125);
```

Additional Example

The following statement sets the end point of the trendline referenced by `Value99` to the current bar's high price:

```
Value1 = TL_SetEnd(Value99, Date, Time, High);
```

TL_SetExtLeft (Reserved Word)

Disclaimer

Trendlines can be extended to the left or right. This reserved word enables you to toggle the trendline between extended to the left and not extended.

```
Value1 = TL_SetExtLeft(Tl_ID, Extend)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline, and `Extend` is a true/false expression that either extends the trendline to the left or not.

`Value1` is any numeric variable or array. You must assign the trendline reserved word to a numeric variable or array so that you can determine whether or not the reserved word performed its operation successfully.

Remarks

When the reserved word performs its operation successfully, a 0 is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

The following turns off the extend left status of the trendline referenced by `Value99` (that is, the trendline will not extend left):

```
Value1 = TL_SetExtLeft(Value99, False);
```

Additional Example

The following statements draw a trendline at the low of a key reversal bar and extend it to the right:

```
Variable: ID(-1);

If Low < Low[1] AND Close > High[1] Then Begin
 ID = TL_New(Date[1], Time[1], Low, Date, Time, Low);
 Value1 = TL_SetExtRight(ID, True);
End;
```

TL_SetExtRight (Reserved Word)

Disclaimer

Trendlines can be extended to the left or right. This reserved word enables you to toggle the trendline between extended to the right and not extended.

```
Value1 = TL_SetExtRight(Tl_ID, Extend)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline, and `Extend` is a true/false expression that either extends the trendline to the right or not.

`Value1` is any numeric variable or array. You must assign the trendline reserved word to a numeric variable or array so that you can determine whether or not the reserved word performed its operation successfully.

Remarks

When the reserved word performs its operation successfully, a 0 is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

The following turns on the extend right status of the trendline referenced by `Value99` (that is, the trendline will extend right):

```
Value1 = TL_SetExtRight(Value99, True);
```

Additional Example

The following statements draw a trendline at the low of a key reversal bar and extend it to the left and right:

```
Variable: ID(-1);

If Low < Low[1] AND Close > High[1] Then Begin
 ID = TL_New(Date[1], Time[1], Low, Date, Time, Low);
 Value1 = TL_SetExtRight(ID, True);
 Value2 = TL_SetExtLeft(ID, True);
End;
```

TL_SetSize (Reserved Word)

Disclaimer

This reserved word changes the thickness of the specified trendline.

```
Value1 = TL_SetSize(Tl_ID, Num)
```

`Tl_ID` is a numeric expression representing the ID number of the trendline, and `Num` is a numeric expression representing the thickness of the trendline, 0 - 6.

`Value1` is any numeric variable or array. You must assign the trendline reserved word to a numeric variable or array so that you can determine whether or not the reserved word performed its operation successfully.

Remarks

Thickness widths range from 0 (the thinnest) to 6 (the thickest).

When the reserved word performs its operation successfully, a 0 is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

The following statement sets the line style of the trendline referenced by `Value99` to the thinnest line style setting:

```
Value1 = TL_SetSize(Value99, 0);
```

TL_SetStyle (Reserved Word)

Disclaimer

This reserved word enables you to modify the style of the specified trendline.

```
Value1 = TL_SetStyle(Tl_ID, Style);
```

`Tl_ID` is a numeric expression representing the ID number of the trendline whose style you want to change, and `Style` is a numeric expression representing the new line style for the trendline.

The possible styles are:

Style Value	Style Reserved Word	Example
1	Tool_Solid	________________
2	Tool_Dashed	___ ___ ___ ___
3	Tool_Dotted	-----------------
4	Tool_Dashed2	_ - _ - _ - _ - -
5	Tool_Dashed3	_.._.._.._...

You can use either the number or the reserved word. The style only applies when the trendline is set to the thinnest size, which is zero (0).

`Value1` is any numeric variable or array. You must assign the trendline reserved word to a numeric variable or array so that you can determine whether or not the reserved word performed its operation successfully.

Remarks

When the reserved word performs its operation successfully, a 0 is returned. When a reserved word cannot perform its operation, it returns an error code. For a list of error codes, see EasyLanguage Drawing Object Error Codes.

Note It is important to remember that if an invalid ID number is used, the reserved word will return a value of -2 and no additional operations will be performed on any trendlines by the trading strategy, analysis technique, or function that generated the error.

Example

The following sets the line style of the trendline referenced by `Value99` to Tool_Dotted (dotted):

```
Value1 = TL_SetStyle(Value99, 3);
```

Style Value
1 Tool_Solid _________________

2 Tool_Dashed - - - - - - -

3 Tool_Dotted etc.

4 Tool_Dashed2 etc.

5 Tool_Dashed3 etc.

To (Reserved Word)

Disclaimer

This word is used as a part of a For statement where the execution values will be increasing to a finishing value.

```
To
```

Remarks

To will always be placed between two arithmetic expressions.

Examples

```
For Value5 = Length To Length + 10 Begin
 {Your Code Here}
End;
```

To is used here to indicate that for each value of Value5 from Length to Length plus 10, the following Begin… End loop will be executed.

```
Variables: Sum(0), Counter(0);
Sum = 0;
For Counter = 0 To Length - 1 Begin
 Sum = Sum + Price[Counter];
End;
```

To is used here to accumulate the variable Sum for each value of Counter from 0 to Length minus one.

Today (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by This Bar.

Remarks

`Today` is used to reference the most current bar, even when analyzing intraday bars.

`Today` is no longer necessary as the following are equivalent:

```
Value1 = Close of Today;
Value1 = Close of This Bar;
Value1 = Close;
```

Tomorrow (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by Next Bar.

Remarks

`Tomorrow` references the next bar, even when analyzing intraday bars.

`Tomorrow` is no longer necessary as the following are equivalent:

```
Buy at Open Tomorrow + Range Stop;
Buy at Open Next Bar + Range Stop;
Buy at Open[-1] + Range Stop;
```

Tool_Black (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the color name Black .

Tool_Blue (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the color name Blue.

Tool_Cyan (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word Cyan.

Tool_DarkBlue (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word DarkBlue.

Tool_DarkBrown (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word DarkBrown.

Tool_DarkCyan (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word DarkCyan.

Tool_DarkGray (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word DarkGray.

Tool_DarkGreen (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word DarkGreen.

Tool_DarkMagenta (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word DarkMagenta.

Tool_DarkRed (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word DarkRed.

Tool_DarkYellow (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word DarkBrown.

Tool_Dashed (Reserved Word)

Disclaimer

Line style used for trendlines.

The possible styles are:

Style Value	Style Reserved Word
1	Tool_Solid
2	Tool_Dashed
3	Tool_Dotted
4	Tool_Dashed2
5	Tool_Dashed3

Example

The following sets the line style of the trendline referenced by `Value99` to `Tool_Dashed`:

```
Value1 = TL_SetStyle(Value99, Tool_Dashed);
```

Tool_Dashed2 (Reserved Word)

Disclaimer

Line style used for trendlines.

The possible styles are:

Style Value	Style Reserved Word
1	Tool_Solid
2	Tool_Dashed
3	Tool_Dotted
4	Tool_Dashed2
5	Tool_Dashed3

Example

The following sets the line style of the trendline referenced by `Value99` to `Tool_Dashed2`:

```
  Value1 = TL_SetStyle(Value99, Tool_Dashed2);
```

Tool_Dashed3 (Reserved Word)

Disclaimer

Line style used for trendlines.

The possible styles are:

Style Value	Style Reserved Word
1	Tool_Solid
2	Tool_Dashed
3	Tool_Dotted
4	Tool_Dashed2
5	Tool_Dashed3

Example

The following sets the line style of the trendline referenced by Value99 Tool_Dashed3:

```
Value1 = TL_SetStyle(Value99, Tool_Dashed3);
```

Tool_Dotted (Reserved Word)

Disclaimer

Line style used for trendlines.

The possible styles are:

Style Value	Style Reserved Word
1	Tool_Solid
2	Tool_Dashed
3	Tool_Dotted
4	Tool_Dashed2
5	Tool_Dashed3

Example

The following sets the line style of the trendline referenced by Value99 Tool_Dotted:

```
Value1 = TL_SetStyle(Value99, Tool_Dotted);
```

Tool_Green (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word Green.

Tool_LightGray (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word LightGray.

Tool_Magenta (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word Magenta.

Tool_Red (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word Red.

Tool_Solid (Reserved Word)

Disclaimer

Line style used for trendlines.

The possible styles are:

Style Value	Style Reserved Word
1	Tool_Solid
2	Tool_Dashed
3	Tool_Dotted
4	Tool_Dashed2
5	Tool_Dashed3

Example

The following sets the line style of the trendline referenced by Tool_Solid:

```
Value1 = TL_SetStyle(Value99, Tool_Solid);
```

Tool_White (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word White.

Tool_Yellow (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word Yellow.

Total (Reserved Word)

Disclaimer

Reserved word used in an Exit statement to specify the number of contracts to exit from a Long or Short entry.

```
Total
```

Remarks

Specifying the number of contracts to exit from a position is not required in an Exit Statement.

`Total` is exclusively used in an Exit condition to specify the number of contracts to exit.

Examples

```
BuyToCover 1 contract total next bar at market;
```

Generates an order to exit only one contract from a short position on the first price of the next bar.

```
Sell 1 contract total next bar at market;
```

Generates an order to exit only one contract from a long position on the first price of the next bar.

Additional Example

```
BuyToCover 5 contracts total next bar at market;
```

Generates an order to exit only five contracts from a long position on the first price of the next bar.

TotalBarsEvenTrades (Reserved Word)

Disclaimer

Returns the total number of bars that elapsed during even trades for all closed-out trades.

```
TotalBarsEvenTrade
```

Examples

`TotalBarsEvemTrade` returns 73 if the number of bars elapsed during 4 even trades were 40, 23, 6 and 4.

`TotalBarsEvenTrade` returns 10 if the number of bars elapsed during 2 even trades were 7, and 3.

TotalBarsLosTrades (Reserved Word)

Disclaimer

Returns the total number of bars that elapsed during losing trades for all closed-out trades.

`TotalBarsLosTrade`

Examples

`TotalBarsLosTrade` returns 73 if the number of bars elapsed during 4 losing trades were 40, 23, 6 and 4.

`TotalBarsLosTrade` returns 10 if the number of bars elapsed during 2 losing trades were 7, and 3.

TotalBarsWinTrades (Reserved Word)

Disclaimer

Returns the total number of bars that elapsed during winning trades for all closed-out trades.

`TotalBarsWinTrade`

Examples

`TotalBarsWinTrade` returns 73 if the number of bars elapsed during 4 winning trades were 40, 23, 6 and 4.

`TotalBarsWinTrade` returns 10 if the number of bars elapsed during 2 winning trades were 7, and 3.

TotalTrades (Reserved Word)

Disclaimer

Returns the total number of closed-out trades.

`TotalTrades`

Examples

`TotalTrades` returns 7 if the number of closed trades is 7.

`TotalTrades` returns 5 if the number of closed trades is 5.

TradeDate (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the EasyLanguage date of the last time any price field of this symbol was updated.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

TradeDateEX (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the Julian date (accurate to the second) of the last time any price field of this symbol was updated.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

TradeDirectionSeries (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

TradeExch (Reserved Word)

Disclaimer

A quote field that returns the trade exchange.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

TradeTime (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the EasyLanguage time of the last time the symbol was updated.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

TradeTimeEX (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the Julian time (accurate to the second) of the last time the symbol was updated.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

TradeVolume (Reserved Word)

Disclaimer

A quote field that returns a numeric expression representing the trade volume of the last trade.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

TrailingStopAmt (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word SetDollarTrailing.

TrailingStopFloor (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word SetPercentTrailing.

TrailingStopPct (Reserved Word)

Disclaimer

This word is retained for backward compatibility. Replaced by the reserved word SetPercentTrailing.

True (Reserved Word)

Disclaimer

This word is used as the value for an input or valid condition.

```
True
```

Remarks

True can only be used in easy language as the value for a True/False input.

True is the value returned by a correct or valid condition such as, 0 > 1.

Examples

```
Input: Test(True);
```

would set the value of an input "Test" to a default value of `True`.

TrueFalse (Reserved Word)

Disclaimer

Reserved word used to define an input that expects a true-false expression passed by value.

```
TrueFalse
```

Remarks

`TrueFalse` can be used for inputs that can be either `TrueFalseSimple` or `TrueFalseSeries`.

Examples

```
   Input: Switch(TrueFalse);
```

declares the constant `Switch` as a true-false value to be used in a function.

```
   Input: Flag(TrueFalse);
```

declares the constant `Flag` as a true-false value to be used in a function.

TrueFalseArray (Reserved Word)

Disclaimer

Reserved word used to define an input that expects a true-false expression passed by value for each array element.

```
InputName(TrueFalseArray);
```

Remarks

A function input is declared as a `TrueFalseArray` when it is passing in a true-false array by value.

Examples

```
   Input: PassedValues[n](TrueFalseArray);
```

indicates that a true-false array is being passed into the function by value through the Input `PassedValues`.

TrueFalseArrayRef (Reserved Word)

Disclaimer

Reserved word used to define an input that expects a true-false array passed by reference.

```
InputName(TrueFalseArrayRef);
```

Remarks

A function input is declared as a `TrueFalseArrayRef` when it is passing in a true-false array by reference.

Examples

```
   Input: PassedValues[n](TrueFalseArrayRef);
```

indicates that a true-false array is being passed into the function by reference through the Input `PassedValues`.

TrueFalseRef (Reserved Word)

Disclaimer

Reserved word used to define an input that expects a true-false expression passed by reference.

```
TrueFalseRef
```

Examples

```
   Input: PassedValue(TrueFalseRef);
```

declares the constant `PassedValue` as a true-false expression to be used in a function.

TrueFalseSeries (Reserved Word)

Disclaimer

Reserved word used to define an input as a true-false value with history.

```
InputName(TrueFalseSeries);
```

Remarks

`TrueFalseSeries` is used for inputs whose values can be referred to historically.

Examples

```
   Input: Switch(TrueFalseSeries);
```

declares the constant `Switch` as a true-false value to be used in a function, where historical values of `Switch` are available.

```
   Input: Flag(TrueFalseSeries);
```

declares the constant `Flag` as a true-false value to be used in a function, where historical values of `Flag` are available.

TrueFalseSimple (Reserved Word)

Disclaimer

Reserved word used to define an input as a true-false value without history available.

```
InputName(TrueFalseSimple);
```

Remarks

`TrueFalseSimple` can not be used for inputs whose values can be referred to historically.

Examples

```
   Input: Switch(TrueFalseSimple);
```

declares the constant `Switch` as a text string value to be used in a function, restricting `Switch` to be a value that does not contain historical values.

```
   Input: Flag(TrueFalseSimple);
```

declares the constant `Flag` as a text string value to be used in a function, restricting `Flag` to be a value that does not contain historical values.

TtlDbt_By_NetAssts (Reserved Word)

Disclaimer

Retained for backward compatibility.

Tuesday (Reserved Word)

Disclaimer

Returns the number 2 as the value for Tuesday.

```
Tuesday
```

Example

```
   Tuesday
```

…returns a value of 2 as the value for Tuesday.

Under (Reserved Word)

Disclaimer

This word is used to check for the direction of a cross between values.

```
Under
```

Remarks

Used in conjunction with the reserved word "`Crosses`" to detect when a value crosses under, or becomes less than another value. A value (`Value1`) crosses under another value (`Value2`) whenever `Value1` is less than `Value2` in the current bar but `Value1` was equal to or greater than `Value2` in the previous bar.

`Under` is a synonym for `Below`.

Examples

```
If Average(Close, 9) Crosses Under the Average(Close, 18) Then...
```

`Under` is used here to determine the direction of the cross of the values two moving averages on the bar under consideration.

```
If Value1 Crosses Under Value2 Then...
```

`Under` is used here to determine the direction of the cross of the variables `Value1` and `Value2` on the bar under consideration.

Underlying (Reserved Word)

Disclaimer

A quote field that returns the specified value from the data feed.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

UnionSess1EndTime (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

Refer to SessionStartTimeMS, SessionStartDayMS, SessionEndTImeMS, SessionEndDayMS for accessing merged session values.

UnionSess1FirstBar (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

Refer to SessionStartTimeMS, SessionStartDayMS, SessionEndTImeMS, SessionEndDayMS for accessing merged session values.

UnionSess1StartTime (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

Refer to SessionStartTimeMS, SessionStartDayMS, SessionEndTImeMS, SessionEndDayMS for accessing merged session values.

UnionSess2EndTime (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

Refer to SessionStartTimeMS, SessionStartDayMS, SessionEndTImeMS, SessionEndDayMS for accessing merged session values.

UnionSess2FirstBar (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

Refer to SessionStartTimeMS, SessionStartDayMS, SessionEndTImeMS, SessionEndDayMS for accessing merged session values.

UnionSess2StartTime (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

Refer to SessionStartTimeMS, SessionStartDayMS, SessionEndTImeMS, SessionEndDayMS for accessing merged session values.

Units (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

UNSIGNED (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

UpperStr (Reserved Word)

Disclaimer

Used to convert a string expression to uppercase letters.

`UpperStr("Str");`

`Str` the string expression to change to uppercase letters.

Remarks

Make sure the string expression `Str` is enclosed in quotation marks.

Example

`UpperStr("trad")` returns the string expression "TRAD".

UpTicks (Reserved Word)

Disclaimer

Returns the number of ticks on a bar whose value is higher than the tick immediately preceding it.

```
Upticks
```

Remarks

Returns the Up volume of a bar when Trade Volume is used for the chart. See EasyLanguage Reserved Words Related to Ticks, Volume & Open Interest for more information.

Examples

`Upticks` returns 5 if there were 5 ticks on a bar whose value was higher than the tick immediately preceding it.

`Upticks` returns 12 if there were 12 ticks on a bar whose value was higher than the tick immediately preceding it.

Additional Example

To check if a bar of data appears to reflect a steady upturn, compare `Upticks` to `DownTicks`:

```
   Value1 = Upticks - DownTicks;
```

V (Reserved Word)

Disclaimer

Shortcut notation that returns the Volume of a bar.

```
V
```

Remarks

Anytime the Volume of a bar is needed, the letter `V` can be used in an equivalent fashion.

Examples

```
   V of 1 bar ago
```

returns the Volume of the previous bar.

```
   Average(V, 10)
```

returns the Average of the last 10 Volume prices.

Additional Example

To check that the last two bars have Volume prices lower than the previous bar write:

```
   If V < V[1] and V[1] < V[2] then
    Plot1(High, "Falling");
```

Value1-99 (Reserved Word)

Disclaimer

This reserved word refers to pre-declared variables in EasyLangauge that can be used to store a value from a calculation or an expression.

A variable is a name that represents a stored value that was assigned from another value or calculation so that you can reference it later in your code. In EasyLanguage, you can think of variables as storage containers that hold a value. This value can be a number or a string (when enclosed in quotes).

Note EasyLanguage provides you with 100 pre-declared numerical variables (Value0 through Value 99) and 100 pre-declared true/false variables (Condition0 through Condition99).

Variable (Reserved Word) same as Var, Vars, Variables

Disclaimer

The reserved word Variable (or Var, Vars, Variables) is used to specify the name of a user-declared variable, its initial value, and optional data type. This must be done before a user-declared variable can be used in an assign statement or formula. Multiple variables names may be declared using a single Variable statement where the names are separated by commas.

Syntax-Common

```
Variable: Name(Value) ;
```

Where Name is the unique name of the user-declared variable, and Value is either a Numeric, True/False or String value used as the initial value of the variable.

Syntax-Complete

```
Variable: <IntraBarPersist> <DataType> VarName(InitialValue<,datan>) <,
<IntraBarPersist> <DataType> VarName(InitialValue<,datan>)> ;
```

- Variable is the variable declaration statement that may alternately be typed as Var, Vars or Variables.
- IntraBarPersist indicates that a variables value can be updated on every tick. By default, variable values are only updated at the close of each bar.
- DataType is the optionally supplied data type (float, double, int, bool, string) of the variable
- VarName is the user specified variable name that can be up to 20 characters long and must begin with an alphabetic character (names are not case sensitive)
- InitialValue is the initial value of the variable
- datan is an optional parameter that identifies the data stream the variable is tied to and may be data1 through data50.

Remarks

A variable name can contain up to 20 alphanumeric characters plus the period (.) and the underscore (_).

A variable name can not start with a number or a period (.).

The default value of a variable is declared by a number in parenthesis after the input name.

Multiple variables names may be declared using a single Variable statement where the names are separated by commas.

The use of the reserved words Variable,Var, Vars, and Variables is identical.

Examples

```
Variable: Count(10);
```

declares the variable Count and initializes the value to ten.

```
Var: MADiff(0), MyAverage(0);
```

declares the variables MADiff and MyAverage and initializes their values to zero.

VARSIZE (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

VARSTARTADDR (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

VEGA (Reserved Word)

Disclaimer

VEGA is used in OptionStation to return the risk value Vega of an option or position.

VOID (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

Volume (Reserved Word)

Disclaimer

Reserved word used to return the Volume of the specified time increment or group of ticks.

```
Volume
```

Remarks

V can be used in place of Volume. See EasyLanguage Reserved Words Related to Ticks, Volume & Open Interest for more information.

Examples

```
   Volume of 1 bar ago
```

returns the Volume of the previous bar.

```
   Average(Volume, 10)
```

returns the Average of the last 10 Volume prices.

Additional Example

To check that the last two bars have Volume prices lower than the previous bar write:

```
   If Volume < Volume[1] and Volume[1] < Volume[2] then
    Plot1(High, "Falling");
```

VSBCOMMENTARY (Reserved Word)

Disclaimer

Reserved for future use.

VWAP (Reserved Word)

Disclaimer

A quote field that returns the VWAP (Volume Weighted Average Price) as calculated throughout the day by the TradeStation data-network.

The VWAP is a measure of the price at which the majority of a given day's trading in a given security took place. It is calculated by adding the dollars traded for the average price of the bar throughout the day ("avgprice" x "number of shares traded" per bar) and dividing by the total shares traded for the day.

Note Quote fields do not reference history. In Chart Analysis, they will only plot a value from the current bar forward.

Was (Reserved Word)

Disclaimer

WAS is a Skip Word that performs no function within your code. It is used within a statement only to make the EasyLanguage code more readable.

Remarks

Skip words in EasyLanguage are not necessary, however, they sometimes make it easier to understand the purpose of the code.

Syntax Coloring in the EasyLanguage PowerEditor is dark green for all skip words by default.

Example

```
If the Close[1] was > than 100 Then
      Alert;
```

The word "was" is not necessary and the code functions the same way regardless of its presence.

Wednesday (Reserved Word)

Disclaimer

Returns the number 3 as the value for Wednesday.

```
Wednesday
```

Example

```
   Wednesday
```

…returns a value of 3 as the value for Wednesday.

While (Reserved Word)

Disclaimer

This word initiates a `While` loop statement.

```
While
```

Remarks

A `While` loop statement defines a set of instructions that are executed until a true/false expression returns false. The number of iterations the `While` loop performs depends on the value returned by the true/false expression following `While`.

Note The true/false expression must eventually return false in order to break the loop. If the expression does not evaluate to false, the `While` loop will continue indefinitely and create an infinite loop.

Examples

```
While Condition1 Begin
 {Your Code Here};
End;
```

`While` is used here to initiate the code contained in the `Begin` … `End` section until `Condition1` returns a value of `False`. If `Condition1` is false, the `While` loop is not executed.

White (Reserved Word)

Disclaimer

Sets the plot color or background color to White.

```
White
```

Remarks

There are 16 colors available in EasyLanguage. For more information, see EasyLanguage Colors and Their Corresponding Numeric Values.

Examples

```
Plot1(Value1, "Test", White)
```

plots `Value1` with the name `Test`, and sets the color of `Plot1` to `White`.

```
TL_SetColor(1, White)
```

sets the color of a TrendLine with a reference number of 1 to `White`.

WORD (Reserved Word)

Disclaimer

Reserved for use with custom DLLs designed for EasyLanguage. Refer to the documentation in the TradeStation Developer's Kit for more information about this and the EasyLanguage Tool Kit Library ELKIT32.DLL.

Year (Reserved Word)

Disclaimer

Returns the corresponding year for the specified calendar date. (1998 = 98, 2001 = 101)

`Year(cDate)`

`cDate` is a numeric expression representing the six or seven digit calendar date in the format YYMMDD or YYYMMDD respectively. (1999 = 99, 2001 = 101)

Examples

`Year(990613)` returns a value of 99 because 990613 represents June 13, 1999.

`Year(1011220)` returns a value of 101 because 1011220 represents December 20, 2001.

YearFromDateTime (Reserved Word)

Disclaimer

Returns the year for the specified DateTime value.

```
YearFromDateTime(dDateTime);
```

Where `dDateTime` is a double-precision decimal expression representing the Julian Date (since 1900) and time (since midnight).

Remarks

If a specific date is entered for the numeric expression, it must be a valid Julian date between January 1, 1900 and February 28, 2150, expressed in DateTime format.

Examples

`Value1 = YearFromDateTime(37564.61200);` returns a value of 2002 for the specified DateTime of 37564.61200 (11/4/2002 2:41:17 PM).

Yellow (Reserved Word)

Disclaimer

Sets the plot color or background color to Yellow.

```
Yellow
```

Remarks

There are 16 colors available in EasyLanguage. For more information, see EasyLanguage Colors and Their Corresponding Numeric Values.

Examples

```
   Plot1(Value1, "Test", Yellow)
```

plots `Value1` with the name `Test`, and sets the color of `Plot1` to `Yellow`.

```
   TL_SetColor(1, Yellow)
```

sets the color of a TrendLine with a reference number of 1 to `Yellow`.

Yesterday (Reserved Word)

Disclaimer

This word is retained for backward compatibility.

Remarks

`Yesterday` references the previous bar, even when analyzing intraday bars.

`Yesterday` is no longer necessary as the following are equivalent:

```
  Value1 = Close of Yesterday;
  Value1 = Close of 1 Bar Ago;
  Value1 = Close[1];
```

Index