TRADE®
NAVIGATOR
BY GENESIS FINANCIAL TECHNOLOGIES

Functions Manual 2012

Genesis
Financial Technologies Inc.

Finally Strategy Development and Back Testing Just Got Easier!

TradeSense™: Functions Manual 2012

Welcome to the powerful world of Genesis Financial Technologies and the Trade Navigator. Since 1984 thousands of market professionals, investors, brokerage firms, and successful traders have relied on the products and services of Genesis Financial Technologies.

Contacting Genesis
For technical support of the Trade Navigator software program please contact:

Genesis Support

Technical Support

Email: Support@TradeNavigator.com
Phone: 719.884.0245
Hours: 6AM - 7PM MT

Sales Support

Email: Sales@TradeNavigator.com
Toll-Free: 800.808.3282
Phone: 719.884.0244
Hours: 8AM - 5PM MT

Mailing Address

Genesis Financial Technologies Inc
4775 Centennial Blvd, Suite 105
Colorado Springs, 80919

Training Support

Email: Training@TradeNavigator.com
Hours: 8AM - 5PM MT

Billing Support

Email: Billing@TradeNavigator.com
Phone: 719.884.0266 Fax: 719.260.6113
Hours: 8AM - 5PM MT

Disclaimer:

Each function, and formula, in this manual is attributed as best as possible. There are some items that were developed by, or for, third party educators for public release. In some cases the educator in question is no longer associated with Genesis Financial Technologies, and has asked that their name be removed. In those cases, the developer of the function, or the idea that function is based upon, has been changed to Educator in order to acknowledge the third party nature of the design or development.

How to Use this Manual:

Below each function name you will find three sub-sections. The main section being the actual name of the function, as well as its definition. Following is a list of each of the three sub-headers that you will find.

Function:

This is what the actual function will look like when referenced. For quick reference all inputs that the function requires are listed within the parenthesis.

Inputs:

This section will define each input that the function depends on. When using TradeSense, most functions require parameters in order to calculate correctly. Instead of leaving the user to wonder what these parameters are requiring from them, we have included this section to provide a clear and consise list of the inputs. In some cases it may also include an example for the input.

Example:

This section provides example uses for each function. Keep in mind that these are just examples and should not be depended on in your trading strategies.

What Are Functions?

Functions are indicators, date/time functions, data functions, and mathematical calculation formulas stored under a one word name (making it easier to reference a formula without writing it out each time you would like to include it). We use functions to specify True/False conditions within entry/exit rules, highlight bars, and we also use them to return number values for indicators.

In this document we will explore the Function component a little further. Following this exploration you will find a Function Glossary, providing explanations, and common uses of all of the available Functions.

Since Functions are such an important part of any advanced/custom charting analysis, Mechanical System creations, and/or Filters/Scans we need to establish a solid understanding of them. Functions are the heart of any trading language, defining certain indicators, data arrays, calendar/time conditions, mathematical formulas, or even simple True/False conditions. Since computers do not understand spoken or even written English, we the software developer, have pre-programmed terms that the computer will understand in a format that they understand. This makes it easier for the user to express conditions in a market environment without having to re-type whatever is included in the function. So, we have created a language called TradeSense™ that is based in English, providing an extensive list of pre-programmed base Functions, allowing for the easy creation of other Functions (making use of the base Functions). Instead of forcing the user to concentrate on, a cumbersome and most often very difficult IF/THEN type C++ programming language, we have created TradeSense™ with all of the tedious programming code done in the background. With all this programming done in the background, it allows users to take full advantage of the Function list, and easily express in English, conditionals or formulas as indicators, highlight bars, and/or entry and exit rules for mechanical systems.

When creating a new Function you will see that after you write your formulas, the Verify button will highlight. *Verify* is a verification process, that will look at your formula and confirm that you have it written in the correct TradeSense™ format. If you have written it incorrectly, it will highlight the incorrect part of the formula in Red. If it is written correctly, you will not see any Red, but it will set the formula in Italics.

You will also notice that you have a Advanced Settings box that you can check. Once this box has an "X" in it, the Function Editor will add a third tab called Advanced. On this tab you can specify what this Function is designed for, Charting, Filters/Scans, or System Testing.

After you verify a Function, you can *Save* it, by clicking the *Save* button at the top and entering a name for your new function.

Following is a comprehensive list of the TradeSense™ functions, including expanded explanations, as well as examples on their common uses.

Operators:

Operators allow users to separate functions, as well as relate them together. By using these operators, it will allow you to input more than one condition into criteria/rules in both the Trade Navigator Gold and our mechanical system back-testing software Trade Navigator Platinum.

Ex: IF Close > Open.1 And MovingAvg (Close, 18) > MovingAvg (Close, 40)

In the example above we are using a “>” sign which is a very commonly used operator within our TradeSense™ language. Also, we used “And” to add another condition to our rule. Following is a list of operators that are available to use within the softwares.

>	This is used when you want one function to be greater than the function following the operator.
>=	This is used when you want one function to be greater than, or at least equal to, the function on the right side of the operator.
<	This is used when you want one function to be less than the function that follows the operator.

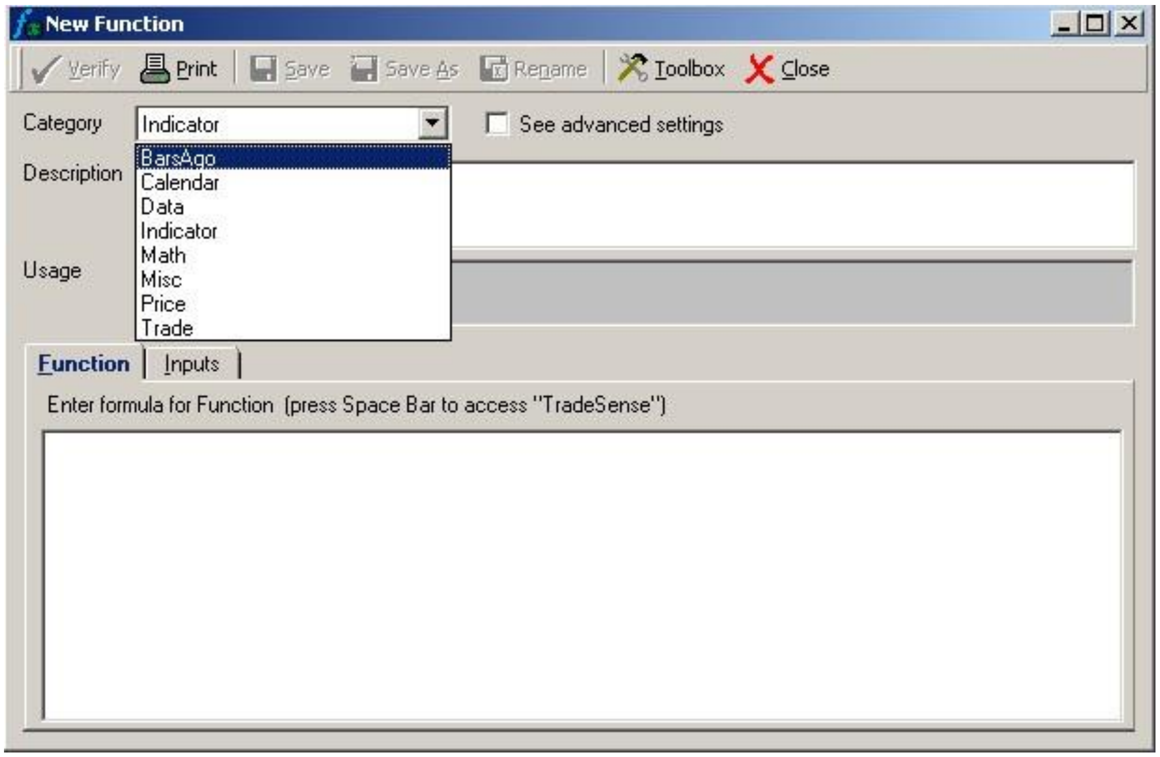
<=	This used when you want one function to be less than, or at least equal to, to the function that follows the operator.
<>	This is used when you do not want a function to equal the function that follows the operator.
=	This is used when you want two functions to equal one another.
+	This is used to add one or more values together.
-	This is used to subtract one or more values from each other.
/	This is used to divide one or more values.
*	The asterick sign allows you to multiply two values.

AND	This is used when you want to have two or more conditions in a criteria/rule to be true before any action takes place. (Ex: IF Close > Open.1 **AND** Open.1 < Open)
OF	This operator is used to link functions with specific data. For example: Close Of GC < Close Of TQ, or ADX (7) Of GC > 60
OR	This is used when you want one, of a number of conditions, to be true before an action takes place. (Ex: IF Monday OR Thursday)
NOT	This is usually used in conjunction with the "AND" operator to specify that you do not want a function to be true. (Ex: IF Monday AND **NOT** Inside Day)

Function Categories:

When exploring Functions, or when creating new Functions, you will run into questions concerning the type of Categories different Functions fall under (see illustration below). We have separated each Function into one of eight different Categories. Below you will find a list of nine Categories you have to choose from (including the *Reserved Category) when sorting your Function list, or creating a new Function.

*The Reserved category is an advanced category that users will not be able to designate within new Functions. Please see the Reserved explanation below for further info.



BarsAgo – This category is for Functions that return either the number of "barsago" or a specific bar in a day.

Calendar – This category is for Functions that reference components of a calendar, such as months, years, and/or trading days.

Data – This category is for Functions that either return a value in a data file for a given security, or are used to reference data in a text file or spread sheet.

Indicator – This category is for Functions that carry the characteristics of an indicator (for example: ADX (7), will return a value and can be plotted on a chart), or Highlight bars (that return True/False or Boolean arrays).

Math – This category is for Functions that carry through mathematical formulas, returning the results of the equation. The Math category uses mathematical equations as the basis for the Function formula.

Misc. – This category is for Functions that utilize combinations of Functions or were never assigned a specific category.

Price – This category is used for Functions that return price values for a given security (open, high, low, close), or prices that are not yet determined, such as Next bar values. Commodity contract specifications, such as Tick Move, and Tick Value will also fall under this Price category.

Trade – This category is used for Functions that are considered trade related (for example: Entry Signal, Traded Today). These functions will not show up on the Function tab of the Traders Toolbox.

Reserved – This category is used internally for Functions that are used within specific rules, or other functions. These functions will not be seen on the Functions menu in the Traders Toolbox, but will show up in the TradeSense™ menu when creating a rule or function.

Abs

This Math function was designed to return the absolute value of a number. The absolute value of a negative number is its positive equivalent.

Function:

Abs (Number)

Inputs:

Number – This is the number that you wish the Abs function to return the absolute value of. Any expression or statement that returns a number value can be used here.

Example:

If you want the Absolute Value of the spread difference between the ProGo Proffesional's and ProGo Public.	*Abs (Spread (ProGo Proffesional (14), ProGo Public(14))*

April

This Calendar function is used to return True if the current price bar is in the month of April, and False if it is not in the month of April. It was intended for use within mechanical system rules, Filter Criteria/Scans and/or custom functions.

Function:

April

Inputs:

None

Example:

Trading signal to occur in April and Close is greater than the previous bar's Close. *IF April and Close > Close.1*

August

This Calendar function is used to return True if the current price bar is in the month of August, otherwise this function will return False. It was intended for use within mechanical system rules, Filter Criteria/Scans and/or custom functions.

Function:

August

Inputs:

None

Example:

Trading signal to occur in August and the Close greater than the previous days Close. *IF August and Close > Close.1* is

---

Average Entry Price

This Price function was designed to average all of the entry prices of the current position, and return the average entry price for the current position. This function is mainly used in rule creation in mechanical systems.

Function:

Average Entry Price

Inputs:

None

Example:

Average Entry Price is greater than the most *IF Average Entry Price > Entry Price* recent Entry Price.

BarOfDay

This Calendar function was designed to return the number of bars into the day for the current bar. It will return “1” for the first bar of the day, “2” for the second bar of the day, etc. This function is most commonly used within intraday strategy rules designating the bar of the day for the current bar without having to specify the actual time for the current bar.

Function:

BarOfDay

Inputs:

None

Example:

Trading signal that will place a trade on the *IF BarOfDay = 1 and Open < Close* second bar of the day, only if the open is less than the close.

Bars Ago Ignoring Inside Bars

This BarsAgo function is used to return the actual number of bars back it takes to find the specified number of non-inside bars (*NumberofNonInsideBars).* For example, if you want to find how many bars it takes to find 3 non-inside bars, the number returned may be 4, or it may be 14, it simply depends on if there are numerous inside bars.

Function:

BarsAgoIgnoringInsideBars (NumberofNonInsideBar)

Inputs:

NumberofNonInsideBars – This is the number of non-inside bars the function will find and return the number of bars that it took to find the specified amount of non-inside bars.

Example:

In this example we to
BarsAgoIgnoringInsideBars (10) <=15 to find 10 non-inside bars within 15 bars.

Bars In Day

This Indicator function is used to return the number of bars in a specific trading day (specified in the *DaysAgo* input). This function is used when you would like to find certain values (in intraday data) a specified amount trading days ago.

Function:

Bars In Day (Days Ago)

Inputs:

Days Ago – This is the number of trading days ago you are specifying for the calculation of Bars in Day function. This number should be expressed as a positive whole number counting back from the current bar. This number must be greater than 0.

Example:

Highest high in *Highest (High, BarsInDay (6)). FirstBarOfDay (5)* the intraday data file 6 days ago.

Bars Left In Day

This Calendar function was designed to return the number of bars after the current bar left in the current trading day. It will return a zero if it is the last bar of the day. This function is most commonly used within intraday strategy stops or end-of-day exit rules.

Function:

Bars Left In Day

Inputs:

None

Example:

We would like to exit a Long position when there are two bars left in the day.

IF *Bars Left In Day = 3*

THEN *Sell 1 contract*

Next Bar Open

Bars Since

This BarsAgo function was designed to return the number of bars it has been since the specified *Occurrence* of the *Condition*, within a specified number of bars (*The Last N Bars*). It is most commonly used within mechanical system rules, as well as Filter Criteria/Scans, charting indicators and custom functions. Its use provides users with an easy way to reference price, and/or indicator values for certain days in which the specified *Condition* is True or otherwise stated.

Function:

Bars Since (Condition, Occurrence, The Last N Bars)

Inputs:

Condition – This input is the pattern and/or market move that we are looking for. The *Bars Since* function returns the number of bars it has been since the specified *Occurrence* of this input (*Condition*).

Occurrence – This input is the actual *Occurrence* of the *Condition* we need to find. To designate the most recent occurrence or last time that the *Condition* happened use the number "1". For all other preceding *Occurrences* use numbers that are greater than "1". For example: When "2" is used for this input, the *Bars Since* function will return the number of bars it has been since the second to last time the *Condition* took place.

The Last N Bars – This input represents the total number of bars that *Bars Since* uses to find the specified *Occurrence* of the *Condition*. In the example below, you see that the "Last N Bars" input is set to 160, meaning that the *Bars Since* function will use 160 bars (from the current bar back) to find the number of bars it has been since the specified *Occurrence* of the *Condition*.

Example:

In this example, we would like the 7 bar ADX value to be greater than 50 and the closing price to the 18 bar simple moving average of the closes.

Bars Since (ADX (7) > 50 And Close > MovingAvg (Close, 18) , 1, 160) <=10

be greater than

OR Since

the *Bars Since* function will return the "number of bars since" our *Conditions*, notice that we have added a "<= 10" following the function.

Bars Since (ADX (7) > 50 And Close > MovingAvg (Close, 18), 1, 160) = True

Now, instead of returning a number value, the "<= 10" will force the *Bars Since* function to return True (if in fact our condition happened within the last 10 bars of data, otherwise it will return False). Since we have created a True/False return, instead of a number return, we can easily use this formula for Filter Criteria/Scan, highlight bars, and/or the conditional statement in mechanical system rules.

Bars Since Entry

This BarsAgo function was designed to return the number of bars since the entry into the current trade (calculation does not factor in the current bar). The *Bars Since Entry* function is most commonly used within exit rules for mechanical systems. This function allows users to define the number of bars it has been since the bar of entry.

Function:

Bars Since Entry

Inputs:

None

Example:

Exit the position the Bars Since your Entry is greater than or equal to a Delay value.	IF *Bars Since Entry >= Delay* when THEN *Sell at Next Bar Market on Close*

Bars Since Exit

This BarsAgo function was designed to return the number of bars since the most recent exit (calculation does not factor in the current bar). The *Bars Since Exit* function is commonly used for entry rules within mechanical strategies. This function allows users to specify the number of bars since the last exit.

Function:

Bars Since Exit

Inputs:

None

Example:

Enter a position, when the bars since the last exit is greater than 3.	IF *Bars Since Exit > 3* THEN *Buy at Next Bar Market on Close*

Bars Since Last Entry

This BarsAgo function was designed to return the number of bars since the most recent entry into the current position. Most commonly used within exit rules when pyramiding is being used. This function will return the number of bars since the last entry of the most recent position.

Function:

Bars Since Last Entry

Inputs:

None

Example:

Exit rule when the bars since the last entry is greater than 3.

IF *Bars Since Last Entry >= 3*

THEN

Sell at Next Bar Market on Close

Bearish Divergence

This Misc function was designed to return True when Bearish Divergence is detected between the *Price* (open, high, low, or close) and the indicator (*Osc*). A bearish market is defined as a market that is trending downward for a long period of time. So, looking for Bearish Divergence (divergence being the key word) between the price and an indicator (*Osc*), the price must be making higher highs, and the indicator (*Osc*) must be making lower highs. Basically, it is when the indicator is failing to match the new highs of the price. Many traders see this as a market top, and may look deeper for a sell signal.

Function:

Bearish Divergence (Price, Osc, Strength)

Inputs:

Price – This is the price that you would like to be making higher highs, detecting Bearish Divergence when compared to the *Osc* function. For example, Open, High, Low or Close are the most commonly used Price inputs.

Osc – This is the indicator that you would like to be making lower highs, detecting Bearish Divergence when compared to the *Price* function.

Strength – This parameter designates the length of the Bearish Divergence. Looking for Bearish Divergence for 10 days, the Strength parameter would be set to 10.

Example:

Current Bearish Divergence between the Close, and the 12/26 MACD indicator for the last 12 bars.	*Bearish Divergence (Close, MACD (Close, 12,26),12)*

Bonds

This Data function returns the closing value for the pit traded 30 Year T-Bonds (TQ). It was designed for use within mechanical systems, providing users the ability to cross reference the Bond market. It can also be referenced within other functions as well as Filter Criteria/Scans.

Function:

Bonds

Inputs:

None

Example:

Bonds yesterday closed higher than they did two days ago.	*Bonds.1 > Bonds.2*

Bullish Divergence

This Misc function was designed to return True when Bullish Divergence is detected between the *Price* (Open, High, Low or Close), and the indicator (*Osc*). A bull market is defined as a market that has been trending up for a long period of time. So, when looking for Bullish Divergence between the price and an indicator, the price must be making lower lows, and the indicator must be making higher lows. Basically, it is when the indicator fails to match the new lows of the price. Many traders see this as a market bottom and will look deeper for a buy signal.

Function:

Bullish Divergence (Price, Osc, Strength)

Inputs:

Price - This is the Price that must be making lower lows when compared to the *Osc*. For example, Open, High, Low or Close are the most commonly used *Price* inputs.

Osc - This is the indicator that must be making higher lows failing to match the new lows of the *Price*.

Strength - This parameter designates the length of the Bullish Divergence. Looking for Bullish Divergence for 10 days, the Strength parameter would be set to 10.

Example:

Current Bullish Divergence between the Close, and the 12/26 MACD indicator for the last 12 bars.	*Bullish Divergence (Close, MACD (Close, 12,26),12)*

Close

This Price function returns the closing price of either Market 1 in your mechanical systems Data tab, or the closing price of a user specified data set (i.e. Close of GC). Within system rules users may specify a separate market by entering an unrecognized input, such as, Close of JOE. The system will recognize "JOE" as a data set (because JOE was inputted after the "Of" operator), and add it to the data tab of the System Editor. When using Close within indicators, the closing price of the charted security is used.

Function:

Close

Inputs:

None

Example:

S&P futures closed than the close of the Gold market.	*Close of SP > Close of GC* higher

Consecutive

This Misc function returns the number of consecutive *Conditions* from the current bar back, to the beginning of the data. Most commonly used within Filter Criteria/Scans, Highlight bars, and/or in rule conditions, specifying that a certain number of conditions have taken place consecutively. In order for this function to be used within Filter Criteria, Highlight bars, and Entry/Exit rules, it must be followed by an operator and the expected value, making it a True/False statement (please see the example below).

Function:

Consecutive (Condition)

Inputs:

Condition – This input is the actual Condition (i.e. open is greater than open yesterday), that has occurred consecutively.

Example:

Five consecutive down closes for either	*Consecutive (Close < Close.1) > 5*

Highlight bars, Entry/Exit rules or a
Filter Criteria/Scan.

Cosine

This Math function returns the cosine of the specified *Number*. Inputs are represented in Radians. This function was designed simply to perform the cosine operation, and may not necessarily be used by the common user. Cosine is most commonly used internally, but is also provided as a common math function.

Function:

Cosine

Inputs:

None

Example:

Cosine is designed to perform the basic cosine operation.

CRB

This Data function returns the closing price for the CRB index futures contract (CR). CRB provides the user with an easy way to cross reference the closing price of the Commodity Research Bureau futures contract (CR).

Function:

CRB

Inputs:

None

Example:

CRB yesterday closed higher than the close of CRB today. *CRB .1 > CRB*

Crosses Above

This Misc function was designed to indicate whether the first input has (*Crossing Value*) crossed above the second input (*Value being Crossed*), by returning a True value. Otherwise this function will return a False value. The formula for the Crosses Above function is as follows: *Crossing Value > Value Being Crossed and Not (Crossing Value > Value Being Crossed).1*

*Please note, that if the two inputs do not return values that are bounded by the same ranges then they may never cross each other. For example, a Stochastic %K indicator will never cross above/below the closing price of a security because, Stochastic %K is a bounded oscillating indicator that always returns values between 0 and 100. Remember that a securities price is unbounded, meaning that they are not expected to return any certain value.

Function:

Crosses Above (Crossing Value, Value Being Crossed)

Inputs:

Crossing Value – This input is the function that is crossing above the *Value Being Crossed*. It can be anything from a standard Stochastic indicator to a price function such as an opening price value.

Value Being Crossed – This input is the function that is being crossed by the *Crossing Value*. It can be anything from a standard Stochastic indicator to a price function such as an opening price value.

Example:

The current close has crossed above the 18 bar moving average of the closes.	*Crosses Above (Close, MovingAvg (Close, 18)*

Crosses Below

This Misc function was designed to indicate whether the first input (*Crossing Value*) has crossed below the second input (*Value Being Crossed*), by returning True. The formula for this function is as follows: *Crossing Value < Value Being Crossed and Not (Crossing Value < Value Being Crossed).1*

*Please note, that if the two inputs do not return values that are bounded by the same ranges then they may never cross each other. For example, a Stochastic %K indicator will never cross above/below the closing price of a security because, Stochastic %K is a bounded Oscillating indicator that always returns values between 0 and 100. Remember that a securities price is unbounded, meaning that they are not expected to return any certain value.

Function:

Crosses Below (Crossing Value, Value Being Crossed)

Inputs:

Crossing Value – This input is the function that is crossing below the *Value Being Crossed*. This input can be anything from a standard Stochastic indicator to a price function such as the Open.

Value Being Crossed – This input is the function that is being crossed by the *Crossing Value*. This input can be anything from a standard Stochastic indicator to a price function such as an opening price value.

Example:

The 18 bar Moving Average of the closes has crossed below the Close.	*Crosses Below (MovingAvg (Close, 18), Close)*

DayOfMonth

This Calendar function returns the day of the month in number format. For example, if today is Wednesday the 2nd of April, this function will return 2. You can also specify the day of the month that you wish (forcing a True/False return) by using the DayOfMonth function combined with the equal (=) operator followed by the day of the month number (see example below).

Function:

DayOfMonth

Inputs:

None

Example:

New Function that Highlights *DayOfMonth = 3* the
3rd of each month on a chart.

Condition to enter a trade next bar, IF *DayOfMonth = 3* if
today is the 3rd of the month.

DayOfWeek

This Calendar function returns the day of the week as a number (1 – 5). For example, if it is Tuesday, this function will return 2. You can also specify the day of the week you want (forcing a True/False return) by using the DayOfWeek function combined with the equal (=) operator followed by the day of the week number (see example below).

Function:

DayOfWeek

Inputs:

None

Example:

New Function that highlights price *DayOfWeek = 3 and Lower (Close, 2)* bars
when it is Wednesday and the close is greater than the close 2 bars
before.

December

This Calendar function returns True if the current bar is in December. Otherwise, this function will return a False value. The December function is mostly used within Entry/Exit rule conditions to limit trading to just days that are in December.

Function:

December

Inputs:

None

Example:

Entry rule that enters a position when the current bar is in December, and Bonds closed (current bar) lower than they did 10 bars ago.	IF *December and Lower (Bonds,10)*

Dollars To Price

This Price function converts the specified dollar amount (*Dollar Amount)*, into the equivilant price move. For example a $500 dollar move in the 30 year T-Bonds (TQ) will be converted to .50. Most commonly used within Entry/Exit rules specifying stop or limit prices, based on a dollar amount (such as Stop Loss or Profit Target rules). The example Long Exit rule below, shows that if the close yesterday was less than the close today, the strategy will exit the long position on a Limit (if the price gets up to a specified price), at the entry price plus $1500. Since many price functions return the actual value of the security (such as Entry Price, Close, etc.), we can not simply add $1500 to them (Entry Price + $1500), the value returned would not be appropriate. Dollar amounts must be converted to price moves in order to be added, subtracted, etc. from a securities price.

Function:

Dollars To Price (Dollar Amount)

Inputs:

Dollar Amount – This is the specific dollar amount to be converted to a price move for the given security.

Example:

An exit rule that liquidates a long position on a Limit at the Entry Price plus $1500, when the current bars close is greater than the previous bars close.	IF *Close > Close.1* THEN *Long Exit on a Limit*

Entry Price + Dollars To Price (1500)

Down Range

This Indicator function returns True if a bar has a lower low, and a lower high than the previous bar. The Down Range function returns False otherwise. Designed to automatically identify Down Range bars within Entry/Exit rules, Filter Criteria/Scans, and/or Charting (indicators/highlight bars). In order for a bar to be considered a Down Range bar it must meet the following condition: *Low < Low.1 and High < High.1*

Function:

Down Range

Inputs:

None

Example:

Close 1 bar ago is less than Close 2 bars ago and the current bar is a Down Range bar. This example formula can be used as a Highlight bar, Filter Criteria, and/or Entry/Exit rule.	*Close.1 < Close.2 and Down Range*

Entry Date

This Reserved function returns the entry date of a specified trade. The Entry Date function is mostly used within Entry/Exit rules when back-testing mechanical strategies.

*The Entry Date function was not designed for use within Filter criteria/Scans, Indicators, Highlight bars, or Custom Functions.

Function:

Entry Date

Inputs:

None

Example:

A rule that liquidates a long position current close is less than the 18 bar moving average of the close, and the entry date	IF *Close < MovingAvg (Close, 18) and* when the *Entry Date*

fell on a Monday. THEN *Long Exit*

At Market

Entry DateTime

This Reserved function returns the date and time of the most recent entry into the current position. The Entry DateTime function was designed for use within Entry/Exit rules of intraday strategies. It allows users to compare the date and time of an entry with other date/time related functions, such as Last ExitDate.

Function:

Entry DateTime

Inputs:

None

Entry Price

This Price function returns the price at which the current trade was entered. Entry Price was designed for use within Exit rules that use Stop or Limit orders (stop losses, profit target rules). This function provides an easy way to reference the price at which the current position was entered. The prices that are returned are for the security being used for back-testing (Market1). Entry Price may be used in the conditional Entry/Exit statement, and/or in the Stop or Limit order price.

Function:

Entry Price

Inputs:

None

Example:

Example use in Stop order price: IF *True* Sell when current price is $1500 less than the entry price.
THEN *Sell on a Stop*

Entry Price – Dollars To Price (1500)

Example use within Exit Condition: IF *Close > Entry Price + Dollars To Price (500)*
Sell at Market price when the current close is $500 above the entry price.
THEN *Sell at MARKET*

Entry Time

This Reserved function was designed to return the time of the entry into the current trade (assuming you are currently in a position).

Function:

Entry Time

Inputs:

None

Exit DateTime

This Reserved function returns the date and time of the most recent exit. The Exit DateTime function was designed for use within Entry/Exit rules of intraday strategies. It allows users to compare the date and time of an exit with other date/time related functions, such as Last EntryDate.

Function:

Exit DateTime

Inputs:

None

Exit Price

This Trade function returns the price at which the last position was exited at. It is most commonly used in entry rules, such as Stop and Reverse orders. Exit Price was designed for use within Entry/Exit rules that use Stop or Limit orders (stop losses, profit target rules). This function provides an easy way to reference the price at which the last position was liquidated. The prices that are returned are for the security being used for back-testing (Market1). Exit Price may be used in the conditional Entry/Exit statement, and/or in the Stop or Limit order price.

Function:

Exit Price

Inputs:

None

Example:

Last exit price is *ExitPrice* < *Close* less than the current close.

Exit Profit

This Trade function returns the profit (or loss) from the last entry to the last exit. Most commonly used to analyze the trade profit of the last position within strategy rules. Exit Profit was designed for use within Entry/Exit rules, and not Filter Criteria/Scans or charting indicators.

Function:

Exit Profit

Inputs:

None

Example:

Most recent exit profit *Exit Profit >= 1000* is greater than or equal to 1000.

Exit Signal

This Trade function was designed to return the rule name of the most recent exit. This function is rarely used, but can be used within Entry rules with the following syntax: IF Exit Signal = “RuleName” THEN…...

*Exit Signal can not be used within Filter Criteria/Scans, or charting indicators/highlight bars.

Function:

Exit Signal

Inputs:

None

False

This Misc function will always return False, and was designed to be used in conjunction with other functions such as Inside bar (see example below). This False function helps when you need a condition to be false inside Entry/Exit rules, Filter Criteria/Scans or advanced charting indicators.

Function:

False

Inputs:

None

Example:

Exit condition where the current is not an inside bar.	IF *InsideBar* = *False* bar
	THEN *Long Exit*
	Market

Feburary

This Calendar function returns True if the date falls within the month of Feburary. Otherwise this function will return False. It has many uses inside Entry/Exit rules (such as seasonal pattern trading) and highlight bars.

Function:

February

Inputs:

None

Example:

Long Entry rule to enter the market only if the current bar is in the month of February, and the entry bar is Friday.	IF *February and Next Bar DayOfWeek = 5* THEN *Long Entry* *NextBarOpen*

FirstBarOfDay

This BarsAgo function returns the number of bars back to the first intraday bar of the specified number of trading days ago. The FirstBarOfDay function is very useful when analyzing price, or indicator values on the first intraday bar of a specific day. Since FirstBarOfDay returns "the number of bars ago", it can be used in conjunction with an "offset" operator. For example, Close.4 will return the closing price of 4 bars ago (the period is considered the "offset" operator). FirstBarOfDay may be used in the same manner, for example, Close.FirstBarOfDay (4) will return the first bar of the days closing price, 4 trading days ago. This function can be used within Filter Criteria/Scans, Entry/Exit rules, and/or Customer functions.

Function:

FirstBarOfDay (Days ago)

Inputs:

Days Ago – This input is the "number of days ago". How many bars has it been since the first intraday bar of this (DaysAgo) trading day? Using "0" will equal today.

Example:

In this example the close of the first bar of the day two trading days ago must have crossed above the 18 bar moving average on the first bar of the day two trading days ago.	*Crosses Above (Close.FirstBarOfDay (2), MovingAvg (Close,18).FirstBarOfDay (2))*

Another example of FirstBarOfDay is a Custom *High. FirstBarOfDay (0)* Function that plots the high of todays first bar.

Floor

This Math function returns the closest whole number below the specified number (for example: Floor (3.5) will return 3.0). The Floor function may be used within Filter Criteria/Scans, Entry/Exit rules, and/or Custom Functions.

Function:

Floor (Number)

Inputs:

Number – User specified value that the Floor function returns the whole number for. This input can be any expression that returns an array of numbers. For example, Floor (High.4), will return the closest whole number below the high 4 bars ago.

Example:

Formula for finding the closest whole *Floor (14.395)* number just under 14.395.

Fractional Part

This Math function returns the fractional part of the specified *Number* (for example, FractionalPart (3.5) will return .5). The Fractional Part function can be used within Filter Criteria/Scans, Entry/Exit rules, and/or Custom Functions.

Function:

Fractional Part (Number)

Inputs:

Number – User specified value that the Fractional Part function returns the fractional part of. This input can be any expression that returns an array of numbers.

Example:

In this example, multiply the current bars close by just the Fractional part of the highest close in the last 10 bars.

*Close * Fractional Part (Highest (Close* , 10))

Friday

This Calendar function was designed to return True if the current bar is Friday. Otherwise this function will return false. The Friday function can be used within Filter Criteria/Scans, Entry/Exit rules, and/or Custom Functions. Friday is most commonly used within Entry/Exit rule conditions in order to generate signals for the next trading day.

Function:

Friday

Inputs:

None

Example:

Highlight bar example, were all bars that are Fridays, and the current bars close is less than the previous bars close are highlighted.

Close < Close.1 and Friday

Gold

This Data function returns the most recent closing price of the Gold market (GC). It’s most common use is cross-referencing the close of Gold within Entry/Exit rule conditions. The CRB Index, and BONDS functions have been added as well to provide an easy way to reference other common markets.

Function:

Gold

Inputs:

None

Example:

True/False statement for Gold closing lower *Gold.1 > Gold* than it closed on the previous bar.

High

This Data function returns the high price for the current bar unless otherwise specified (For example: High.2 will return the High of 2 bars ago). The *High* function was intended to be used in Filter Criteria/Scans, Custom Functions, as well as Entry/Exit rules. When used within a Custom Function formula, it defaults to the high of the security that is charted. When used in a Entry/Exit rules, it will default to the High price of Market1 (found on the Data tab of the System Editor).

Function:

High

Inputs:

None

Example:

In this example, if the High of yesterday *High.1 > MovingAvg (High, 10).1* was greater than yesterdays 10 bar Moving Average of the highs, then highlight the bars that carry a True value for the condition.

Higher

This Misc function returns True if the specified Expression has a Higher value than the same Expression a certain number of traded bars ago (*Compared to N bars Ago*). The Higher function compares only two values, the Expressions current value and the Expressions value “N” bars ago. Higher is mostly used within Filter Criteria/Scans, Entry/Exit rules, and/or Custom Functions.

Function:

Higher (Expression, Compared to N bars ago)

Inputs:

Expression – The "value" that you would like to be higher than it was "N" bars ago. This "value" can be anything (indicator, price, etc.) that returns a number value.

Compared to N Bars Ago – This is the comparison bar for the *Expression* (for example: Using a "2" will compare the *Expression* to that very same *Expression* 2 bars ago).

Example:

In this example, if today's High is greater than the High two bars ago, then return True.	*Higher (High,2)*

Highest

This Indicator function returns the Expression's highest value within a certain period of time (or range of bars) (also user specified; *The Last N bars*). The difference between *Highest* and *Higher*, is that using the *Higher* function will compare the Expression to the same Expression on a certain day in the past (returning True, if the current Expression is Higher than it was on that particular day in the past). When using the *Highest* function, it will return the Highest value (Expression) in a range of past bars. You can use this function in Entry/Exit rules, as well as Filter Criteria/Scans and Custom Functions.

Function:

Highest (Expression, The Last N bars)

Inputs:

Expression – Greatest price or indicator value within *The Last N bars*. The Expressions value will be returned when using the Highest function.

The Last N bars – Number of bars (from the current bar back) that Highest compares, and returns the greatest value.

Example:

Close today is greater than yesterdays close, and the current 7 bar ADX is the highest it has been in the last 12 bars.	*Close > Close.1 and ADX (7) >= Highest (ADX(7) , 12).1*

HighestAt

This BarsAgo function was designed to return the number of bars since, a specified occurrence (i.e. most recent occurrence, 2nd to last occurrence), of the Highest *Expression*, in the specified amount of bars. It is commonly used to find the highest, most recent match, of a value (Indicator, Price function) in a specified length of time.

Function:

HighestAt (Expression, The LastN bars, UseMostRecentMatch)

Inputs:

Expression – This is the value or condition that you need to find the number of bars since, the most recent time (*UseMostRecentMatch*) it was the Highest (of its own values) in the specified amount of bars (*TheLastNBars*).

TheLastNBars – This is the total number of bars that the *HighestAt* function is using to find the number of bars back to the highest, most recent occurrence of the *Expression*.

UseMostRecentMatch – This input is to designate the specific occurrence of the Highest Expression. Using "1" will represent the most recent occurrence, and using "2" will represent the second to last occurrence.

Example:

In this example we are looking for the number of bars since the most recent occurrence of the highest ADX (7 bars) value in 200 bars to be less than 10 bars ago.

HighestAt (ADX (7), 200, 1) <= 10

HighestHigh

This Data function was designed to return the highest high value in the last "N" bars (for filter/filter criteria, custom indicators, or rule creation).

Function:

Highest High (The Last n Bars)

Inputs:

TheLastNBars – This input is the number of bars that you are looking for the Highest High in. For example, if you have 50 in this input, the program will look through the last 50 bars (whether Daily, Weekly, etc) for the HighestHigh value.

Example:

In this example we are specifying that the high value yesterday is greater than the highest high value in the last 50 bars.

High.1 > HighestHigh (50).1

HighestHighInTrade

This Price function was designed to return the highest price that has occurred since the current position was entered. It is intended to be used within mechanical system rules, but not Filter Criteria/Scans or other functions.

Function:

HighestHighInTrade

Inputs:

None

Example:

If you want to exit a position if the Highest High in the current trade has been greater than or equal to your entry price plus 20% of a tick.

*IF HighestHighInTrade >= EntryPrice + TickMove *.20*
THEN
At Market

Hour

This Calendar function was designed to return the current bar hour in 24 hour format. For example: 2:00 p.m. = 14. This function can be used within custom functions (such as highlight bars), as well as mechanical system rules.

Function:

Hour

Inputs:

None

Example:

In this example we would like a highlight bar to highlight all of the 2:00 p.m. bars.

Hour = 14

IFF

This Misc function is an embedded "IF" function that works like an IF/THEN statement. If input 1 (*Condition*) is True, then return value of Input 2 (*Value if Condition is True*), otherwise return value of Input 3 (*Value if Condition is False*). This function is most commonly used in the creation of other custom functions.

Function:

IFF (Condition, Value if condition is True, Value if condition is False)

Inputs:

Condition – This is the condition or statement to be evaluated, expecting the return of either Input 2 (If the condition is True) or Input 3 (If the condition is False).

Value if Condition is True – This inputs value is returned if the *Condition* is in fact True. This input can be anything from standard indicators to price functions, such as the current high price of a security.

Value if Condition is False – This inputs value is returned if the *Condition* is False. This input can also be anything from standard indicators to price functions, such as the current high price of a security.

Example:

In this example, if the closing price was greater than the opening price of the trading day, highlight the highest price bar of the day (on an intraday price chart). If it closed below the opening price, highlight the lowest price bar of that day.

IFF (Close > Open, Close, Open)

Inside Bar

This Data function was designed to return True if a bar did not trade higher nor lower than the previous bar. Otherwise this function will return a False value. The highs and lows can be equal and still remain an Inside Bar.

Function:

InsideBar

Inputs:

None

Example:

In this example, we want the closing price two bars ago to be greater than the closing price of three bars ago, and we also want the last bar to be an inside bar.

Close.2 > Close.3 and InsideBar.1

IntegerPart

This Math function was designed to return the integer part (or whole number) of the specified *Number*. For example, the IntegerPart of 3.5 is 3.

Function:

IntergerPart (Number)

Inputs:

Number – This input is the number you wish the program to convert to just the integer part (or whole number). Remember, that for the function to convert a fractional number to the integer part, this input must be a fractional number.

Example:

In this example, we want to multiply the close of yesterday by the integer part value of the 18 bar moving average of the close.	*Close.1* IntegerPart (MovingAvg (Close, 18))*

IsCommodity

This Misc function was designed to return True, if in fact the security is a future/commodity. This function was intended to be used in the creation of Filters, Filter Criteria, and/or custom functions. It was designed before you could specify the symbol group you wished to calculate the Filter Criteria/Scans on, making it easier to specify that you only wanted to calculate on Commodities.

Function:

IsCommodity

Inputs:

None

Example:

In this example, we want a Filter Criteria/Scan to recalculate and find all of the commodities with a 7 bar ADX value of 60 or above.	*ADX (7) >= 60 and IsCommodity*

IsUndefined

This Misc function was designed to return True if the specified *Expression* is a Null value (or has not happened yet). This can be used within mechanical system rules, Filter Criteria/Scans, and/or charting indicators.

Function:

IsUndefined (Expression)

Inputs:

Expression – This input is the actual value you wish to be Null or non-existent (as of yet).

Example:

In this example, we want to exit a current long position only if the bars since your entry is greater than 5 and the next trading day is Friday (but next trading is yet to be defined).	*IF BarsSinceEntry >= 5 and Isundefined (NextBarDayIs (“Friday”))*

January

This Calendar function was designed to return True if the current price bar is in the month of January. Otherwise this function will return False.

Function:

January

Inputs:

None

Example:

In this example, we want the next trading day to fall in the month of January and the close today is less than the close yesterday.	*NextBarMonth = January and Close < Close.1*

July

This Calendar function was designed to return True if the current price bar is in the month of July. Otherwise, this function will return False.

Function:

July

Inputs:

None

Example:

In this example, we want the next trading day to fall in the month of July and the close today is less than the close yesterday.

NextBarMonth = July and Close < Close.1

June

This Calendar function was designed to return True if the current price bar is in the month of June. Otherwise, this function will return False.

Function:

June

Inputs:

None

Example:

In this example, we want the next trading day to fall in the month of June and the close today is less than the close yesterday.

NextBarMonth = June and Close < Close.1

LastEntryDateTime

This Reserved/Trade function was designed to return the date and time of the most recent entry into the current position. This function was only intended for use in mechanical system rules.

Function: *LastEntryDateTime*

Inputs:

None

Example:

In this example, we want to exit the current position only if the entry date and time came before the third bar of the day (using intraday data).

IF LastEntryDateTime < (BarofDay = 3)

THEN Exit Long

At Market

LastEntryPrice

This Reserved/Trade function was designed to return the price that the most recent entry was filled at. This function was only intended to be used in mechanical system rules, since there would be no use for it in a Filter Criteria/Scan or charting indicators.

Function: *LastEntryPrice*

Inputs:

None

Example:

In this example, we want to exit a position only If the last entry price was less than Yesterdays close.

IF LastEntryPrice < Close.1

THEN Exit Long

At Market

LastEntryProfit

This Reserved/Trade function was designed to calculate the profit/loss from the most recent entry to the last bar of data in the data file and return the profit/loss dollar amount. Automatically assuming you are exiting at the closing price of the current bar, it returns the profit/loss for the complete trade. It was only intended to be used in mechanical system rules, not Filter Criteria/Scans or custom functions.

Function: *LastEntryProfit*

Inputs:

None

Example:

In this example, we want to exit a trade when the entry Profit has exceeded your entry price plus $500.

IF LastEntryProfit > (EntryPrice + DollarsToPrice (500))

THEN Long Exit

At Market

LastEntrySignal

This Reserved/Trade function was designed to return the rule name for the most recent entry into the current position. This function was only intended to be used in mechanical systems rules, not Filter Criteria/Scans or custom indicators.

Function: *LastEntrySignal*

Inputs:

None

LastBarofDay

This BarsAgo function was designed to return the number of bars since the last bar of a specified number of trading days ago. You can also use this for charting indicators, such as Moving averages, of the number of bars back to the last bar of the day (using a specified amount of days ago). This function was intended to be used if the user has subscribed to Genesiss' intraday data.

Function:

LastBarofDay (Days Ago)

Inputs:

Days Ago - This is the number of "Trading Days ago" you need to find the number of bars back to the last bar of that day. Using "0" will equal today.

Example:

In this example, we want to construct a moving average of the closes using the number of bars that it has been since the last bar of yesterdays data. *MovingAvg (Close, LastBarOfDay (1))*

Log

This Math function was designed to return the natural logarithm of a number. The *Log* function was intended to be used in either mechanical system rules, Filter Criteria/Scans, charting indicators, and/or custom functions.

Function:

Log (Number)

Inputs:

Number – This input is the particular number (or any function that returns a number value) that you need the natural logarithm for.

Example:

In this example, we want the logarithm for yesterdays 7 bar ADX value. *Log (ADX (7).1)*

Lower

This Misc function was designed to return True if the current *Expressions* value is lower than the same *Expressions* value on a certain traded bar. Otherwise this function will return False. It simply compares two of the same values on two different traded bars. It can be used within mechanical system rules, Filter Criteria/Scans, charting indicators, and/or custom functions.

Function:

Lower(Expression, Compared to N bars Ago)

Inputs:

Expression – This is the condition that you want to be lower than the same condition "N" bars ago.

Compared to N bars Ago – This is the number of bars back to compare to.

Example:

In this example, we want the current 7 bar ADX value to be a lower value than it was 10 bars before. *Lower (ADX (7), 10)*

LowestAt

This BarsAgo function was designed to return the number of bars since a specified occurrence (i.e. most recent occurrence, 2nd to last occurrence) of the Lowest *Expression* in the specified amount of bars. It is commonly used to find the most recent match of the *Expression*'s Lowest value (Indicator, Price function) in the specified length of time.

Function:

LowestAt (Expression, TheLastN bars, UseMostRecentMatch)

Inputs:

Expression – This is the value or condition that we need to find the number of bars since, the specified occurrence (*UseMostRecentMatch*) of its Lowest value in the specified amount of time (*TheLastNBars*).

TheLastNBars – This is the total number of bars that the *LowestAt* function will use to find, the number of bars since the specified occurrence of its lowest value.

UseMostRecentMatch – This input is to designate the specific occurrence we are using to find the number of bars since (occurrence). Using "1" will represent the most recent occurrence, and using "2" will represent the second to last occurrence.

Example:

In this example, we are looking for the number of bars since the most recent occurrence of the lowest 7 bar ADX value in the last 200 bars to be less than 10 bars ago. *LowestAt (ADX (7), 200, 1) <= 10*

LowestLowinTrade

This Reserved/Trade function was designed to return the lowest price (of Market 1) that has occurred since the current position was entered (assuming there is an existing open trade). This function was only intended to be used within mechanical system rules, since this function returns the lowest low price since a trade was entered. It will not work within Filter Criteria/Scans, or custom functions (such as indicators or highlight bars).

Function:

LowestLowinTrade

Inputs:

None

Example:

In this example we want *IF True* a
stop loss rule that stops
us out at the lowestl low price *THEN Exit Long* in
the last most recent trade.
At LowestLowInTrade

March

This Calendar function is designed to True if the current price bar is in the month of March. Otherwise, this function will return False.

Function:

March

Inputs:

None

Example:

In this example, we want to exit a long *IF March and Close > Close.1* position
only if it is currently March
and the most recent closing price is less than *THEN Long Exit* yesterdays
closing price (assuming the most recent
close was today's close). *At Market*

MarketPosition

This Reserved/Trade function was designed to return the type of position (0 = None, 1 = Long, and -1 = Short) that the current trade is.

Function: *MarketPosition*

Inputs:

None

Example:

In this example, we would *MarketPosition = 1 or MarketPosition = 0* like our current market postion to be a long one or none at all.

Max

This Math function was designed to compare the two specified inputs, and return the greater (or highest) of the two values.

Function:

Max (Comparison Value 1, Comparison Value 2)

Inputs:

Comparison Value 1 – This is the first of two values to be compared.

Comparison Value 2 – This is the second of the two values to be compared.

Example:

In this example, we want the greatest value *Max (Lowest (Close, 3), Close.10) < Max* of either the lowest close of the last 3 bars, or *(Lowest (Close, 18), Close.30)* the closing price 10 bars ago to be less than the greatest value of either the lowest close of the last 18 bars or the close price of 30 bars ago.

MaxPositionLoss

This Reserved/Trade function was designed to return the Maximum Loss (per unit) since the current trade was entered. It was designed to be used in mechanical system rules only, not Filter Criteria/Scans, charting indicators or custom functions.

Function:

MaxPositionLoss

Inputs:

None

Example:

In this example, we would like our maximum position profit to be greater than our maximum position loss.

MaxPositionLoss < MaxPositionProfit

MaxPositionProfit

This Reserved/Trade function was designed to return the Maximum Profit (per unit) since the current trade was entered. It was designed to be used in mechanical systems, not Filter Criteria/Scans, charting indicators or custom functions.

Function:

MaxPositionProfit

Inputs:

None

Example:

In this example, we would like our maximum position profit to be greater than our maximum position loss.

MaxPositionLoss < MaxPositionProfit

May

This Calendar function was designed to return True if the current bar is in the month of May. Otherwise, this function will return a False value.

Function:

May

Inputs:

None

Example:

In this example, we want to specify that the next bar month is in the month of May.

NextBarMonth = May

Min

This Math function was designed to compare the two inputs and return the lesser or lower value of the two.

Function:

Min (Comparison Value 1, Comparison Value 2)

Inputs:

Comparison Value 1 – This input is the first of the two values to be compared.

Comparison Value 2 – This input is the second of the two values to be compared.

Example:

In this example, we want the smallest value of either the lowest close of the last 3 bars, or the closing price 10 bars ago to be less than the lesser value of either the lowest close of the last 18 bars or the close price of 30 bars ago.

Min (Lowest (Close, 3), Close.10) < Min (Lowest (Close, 18), Close.30)

MinMove

This Price function was designed to return the minimum value a security can move (usually it is one increment). For example T-Bonds move 1 tick at a time. This function was designed to work only with mechanical systems and their rules.

Function:

MinMove

Inputs:

None

Example:

In this example, we would like the to exit a position using a stop loss rule at our entry price minus the minimum move of the given security (Market 1).

IF True

THEN Long Exit on a Stop

Entry Price - MinMove

Minute

This Calendar function was designed to return the minute of the current bar in a standard MM format. It is commonly used within mechanical system rules, as well as charting indicators and custom functions that need to reference intraday data.

Function:

Minute

Inputs:

None

Example:

In this example, we would like to *Minute = 30*
have a highlight bar that highlights all of bars on an intraday chart that have a minute value of :30.

MinutesToTime

This Calendar function was designed to convert the minutes since midnight, to a hour/minute format (HHMM). For example 180 minutes since midnight, this function would return 0300 (for three hours or 3:00 a.m.).

Function:

MinutesToTime (Minutes Since Midnight)

Inputs:

Minutes Since Midnight – This input is the minutes since midnight to be converted to a standard HHMM format. In the example above, we use 180 as the *Minutes Since Midnight*.

MinutesPerBar

This Calendar function was designed to return the number of minutes in each bar of the data. If the data is not in a “Minute bars” or Intraday format, then this MinutesPerBar function will return a “0” value. It was designed to be used within mechanical system rules, Filter Criteria/Scans, as well as charting indicators and custom functions.

Function:

MinutesPerBar

Inputs:

None

Example:

In this example, if the chart is set to 60 minute bars, then we would like a highlight bar that highlights all of the bars that have a :30. If the chart is not set to 60 minute bars, do not highlight anything.

IFF (MinutesPerBar = 60, Minute = 30, False) = True

Monday

This Calendar function was designed to return True if the current price bar is a Monday. Otherwise this function will return False. It can be used within mechanical system rules, as well as charting indicators and custom functions.

Function:

Monday

Inputs:

None

Example:

In this example, we would like to enter long positions only on Thursdays when the close on Wednesday is greater than the close on Tuesday (assuming none of the days fall on a holiday).

IF Monday and Close > Close.1
THEN Long Entry
At Market

Month

This Calendar function was designed to return the specific month for the current or most recent bar. Each month is in number format. For example January = 1, February = 2, etc.

Function:

Month

Inputs:

None

Example:

In this example, we would like to highlight all bars that fall in the month of October and have a StochK value of 80 or above.

Month = 10 and StochK (14,3) >= 80

MovingAvg CrossesAbove

This Misc function was designed to return True if the fast moving average of the closes has crossed above the slow moving average of the closes. This function will return False if the *FastMAbars* has not crossed above the *SlowMAbars*. The *MovingAvg CrossesAbove* function was intended for use in Filter Criteria/Scans, mechanical system rules, as well as custom functions that expect a True/False return (such as highlight bars).

Function:

MovingAvgCrossesAbove (FastMAbars, SlowMAbars)

Inputs:

FastMAbars – This input specifies the number of bars used in the average for the fast moving average (moving average that crosses). The default is set to 0.

SlowMAbars – This input is the number of bars used in the average for the slow moving average (moving average that is crossed). The default for this one is also 0.

Example:

In this example, we would like those price bars highlighted that correspond to the 7 bar FastMA crossing above the 40 bar SlowMA.

MovingAvg CrossesAbove (7, 40)

MovingAvg CrossesBelow

This Misc function was designed to return True if the fast moving average of the closes has crossed below the slow moving average of the closes. This function will return False if the *FastMAbars* has not crossed below the *SlowMAbars*. The *MovingAvg CrossesBelow* function was intended to be used in Filter Criteria/Scans, mechanical system rules, as well as custom functions that expect a True/False return (such as highlight bars).

Function:

MovingAvg CrossesBelow (FastMAbars, SlowMAbars)

Inputs:

FastMAbars – This input specifies the number of bars used in the average for the fast moving average (moving average that crosses). The default is set to 0.

SlowMAbars – This input is the number of bars used in the average for the slow moving average (moving average that is crossed). The default for this one is also 0.

Example:

In this example, we would like those price bars highlighted that correspond to the 7 bar FastMA crossing below the 40 bar SlowMA.

MovingAvg CrossesBelow (7, 40)

NextBar BarofDay

This Calendar function was designed to return the number of bars into the current day for the next bar. For example, for the first bar it will return a 1, and for the second bar it will return 2, etc. It was only intended to be used within mechanical system rules.

Function:

NextBar BarofDay

Inputs:

None

Example:

In this example, we would like to enter a position only if the entry bar is after the 3rd bar of the day.

IF NextBarBarofDay > 3

THEN Long Entry

At Market

NextBarClose

This Price function was designed to return the closing price of the very next price bar. This particular function was only intended for use within mechanical system rules, providing the ability to reference the next bar close. It is most commonly used when specifying the price at which to enter or exit a position.

Function:

NextBarClose

Inputs:

None

Example:

In this example, we would like to exit an current position on the next bar close only if today's close is greater than yesterday's close.

IF Close > Close.1
THEN Long Exit
At NextBarClose

NextBarDayIs

This Calendar function was designed to return True, if the next tradable bar is the specified day of the week. If the next tradable bar is not the specified day of the week, the function will return False. It was only intended to be used within mechanical system rules, providing the ability to specify the day of the week for the next bar as part of the condition.

Function:

NextBarDayIs (WeekDayName)

Inputs:

WeekDayName – This input is the actual name of the weekday (Monday, Tuesday, etc.) that must be specified. There is a little quirk with this input. The weekday name must be enclosed in quotation marks.

Example:

In this example, we would like to specify, as the signal for an exit, that the next day of the week is Thursday. Also, we would like the current bar to close above our Entry Price.

IF NextBarDayIs ("Thursday") And
EntryPrice < Close
THEN Long Exit
At Market

NextBar DayOfMonth

This Calendar function was designed to return the day of the month in numeric format for the next bar. This function was only intended for use within mechanical system rules, providing the ability to specify the day of the month for the next bar.

Function:

NextBar DayOfMonth

Inputs:

None

Example:

In this example, we would like to specify, as part of the signal for an entry, that the day of the month for the next bar is the 20th of the month or before.	*IF NextBar DayOfMonth <= 20*

NextBar DayOfWeek

This Calendar function was designed to return the day of the week number for the next bar. For example, 1 = Monday, 2 = Tuesday, etc. Just like all of the other "Next Bar" Calendar functions, this function was only intended to be used within mechanical system rules.

Function:

NextBar DayOfWeek

Inputs:

None

Example:

In this example, we would like the next bar to be the 3rd day of the week, as part of the condition or signal to enter/exit a position.	*IF NextBar DayOfWeek = 3*

NextBarHigh

This Price function was designed to return the price of the next bars high. This function, like the many other "Next Bar" functions was only intended for use within mechanical system rules. Since all orders within system rules are "Next Bar" orders, "Next Bar" functions provide an easy way to specify a price that has not happened yet.

Function:

NextBarHigh

Inputs:

None

Example:

In this example, we would like enter a short position at the next bar's closing price, only if next bar's high value exceeds the current bar's closing value.	*IF NextBarHigh > Close* to *THEN Short Entry* the *At Next Bar Close*

NextBarHour

This Calendar function was designed to return the hour of the next bar (in 24 hour format). For example, 2:00 p.m. = 14. This "Next Bar" Calendar function was only intended for use within mechanical system rules. It is most commonly used as part of the condition (or signal) for entry or exit into a position.

Function:

NextBarHour

Inputs:

None

Example:

In this example, part of the condition for entry into a position is that it must be 11:00 a.m. or after. *IF NextBarHour >= 11*

NextBarLow

This Price function was designed to return the price of the next bars low. This function, like the many other "Next Bar" functions was only intended for use within mechanical system rules. Since all orders within system rules are "Next Bar" orders, "Next Bar" functions provide an easy way to specify a price that has not happened yet.

Function:

NextBarLow

Inputs:

None

Example:

In this example, we would like enter a short position at the next bar's close value, only if the next bar's low value exceeds the next bar's opening value.

IF NextBarLow > NextBarOpen to

THEN Short Entry

At Next Bar Close

NextBarMinute

This Calendar function was designed to return the minute value of time for the next bar. This function returns the minute value based on the data intervals of Market 1. The "NextBarMinute" function was only intended for use within mechanical system rules, returning the minute value for the next bar of data, based on the intraday data interval Market 1 is set to (10 Minute bars, 30 Minutes, etc).

Function:

NextBarMinute

Inputs:

None

Example:

In this example, part of the condition for entry into a position is that the entry bar must be the 11:30 bar.

IF NextBarHour = 11 And NextBarMinute = 30

NextBarMonth

This Calendar function was designed to return the month value for the next bar in number format, where as each number (1 – 12), corresponds to that particular month. For example: NextBarMonth = 3, will return True if in fact the next bars date is in the month of March. The NextBarMonth function, just like many of the other "Next Bar" functions, was only intended for use within mechanical system rules. It is also used in the creation of custom functions, but this usage is rarely seen. Since this function returns the value (expressed as a number) of the Next Bars month when used alone, it is usually followed by a math operator (<, >, =), that is followed by the month number. This process will change the return type, from a number value return, to a True/False return.

Function:

NextBarMonth

Inputs:

None

Example:

In this example, we would like the very next bar to be in the month of February.

IF NextBarMonth = 2

NextBarOpen

This Price function was designed to return the price of the next bars open. This function, like the many other "Next Bar" functions was only intended for use within mechanical system rules. Since all orders within system rules are "Next Bar" orders, "Next Bar" functions provide an easy way to specify a price that has not happened yet.

Function:

NextBarOpen

Inputs:

None

Example:

In this example, we would like enter a short position at the next bar's opening price only if current bar's close value exceeded the current bar's low value.

IF Close > Low to

THEN Short Entry the

At Next Bar Open

NextBarTime

This Calendar function was designed to return the time of the next bar as a number (HHMM). For example 930 is 9:30 a.m., and 1600 for 4:00 p.m. This function, like the many other "NextBar" functions, was only intended for use within mechanical system rules. It is most commonly used as part of the condition or signal, triggering a trade.

Function:

NextBarTime

Inputs:

None

Example:

In this example, we would like a trade to be triggered only if it is after 9:30 a.m.

IF NextBarTime > 930

NextBarTradingDayOfMonth

This Calendar function was designed to return the trading day of the month number for the next bar. For example, if today is 12/31/99, this function will return a 1. This function will automatically skip holidays and weekends. It was also only intended for use within mechanical system rules.

Function:

NextBarTradingDayofMonth

Inputs:

None

Example:

In this example, we want to place a trade on the next bar, but only if it is not the same trading day of the month as the current bar.

IF TradingDayOfMonth <> NextBarTradingDayOfMonth

NextBarTradingDayOfWeek

This Calendar function was designed to return the trading day of the week number for the next bar. For example: Monday = 1, Tuesday = 2, Wednesday = 3, etc. This function was only intended to be used within mechanical system rules.

Function:

NextBarTradingDayOfWeek

Inputs:

None

Example:

In this example, we would like a trade to be triggered only if the next bar falls on any day of the week but Wednesday.

IF NextBarTradingDayOfWeek <> 3

NextBarTradingDayOfYear

This Calendar function was designed to return the trading day of the year as a number for the next bar, skipping weekends and holidays. As with many of the "Next Bar" functions, this function was only intended to be used within mechanical system rules.

Function:

NextBarTradingDayOfYear

Inputs:

None

Example:

In this example, we would like *Close.2 > Close.1 and TradingDayOfYear =*
a trade to be triggered only if *NextBarTradingDayOfYear* the next bars
TDOY is equal to the current TDOY, and the close two bars ago is greater than the close one bar
ago.

NextBarTradingDaysAfterHoliday

This Calendar function was designed to return the number of trading days that have gone by since the last holiday, starting from the last holiday and ending at the next bar. This function, like many of the other "Next Bar" functions, was only intended to be used within mechanical system rules. This NextBarTradingDaysAfterHoliday function makes it easier to specify holiday trading signals.

Function:

NextBarTradingDaysAfterHoliday

Inputs:

None

Example:

In this example, we want to *IF NextBarTradingDaysAfterHoliday = 1*
have a signal or condition that triggers a trade the day after the most recent holiday.

NextBarTradingDaysLeftBeforeHoliday

This Calendar function was designed to return the number of trading days from the next bar to the next holiday. This function, like many of the other "Next Bar" functions, was only intended to be used within mechanical system rules. This NextBarTradingDaysLeftBeforeHoliday function makes it easier to specify holiday trading signals as conditions in system rules.

Function:

NextBarTradingDaysLeftBeforeHoliday

Inputs:

None

Example:

In this example, we want to have a signal or condition that triggers an exit on close one day before the next holiday.

IF NextBarTradingDaysLeftBeforeHoliday = 0

THEN Long Exit

Market on Close

NextBarTradingDaysLeftInMonth

This Calendar function was designed to return the number of trading days left in the current month, starting from and including the next bar. This will return a 1 if it is the last trading day of the month. This function, like many of the other "Next Bar" functions, was only intended to be used within mechanical system rules. This NextBarTradingDaysLeftInMonth function makes it easier to specify conditions that trigger trades when there are a certain amount of trading days left in the month.

Function:

NextBarTradingDaysLeftInMonth

Inputs:

None

Example:

In this example, we would like to enter a position if the next daily bar is the last trading day in the current month.

IF NextBarTradingDaysLeftInMonth = 1

NextBarWeekDayOccurrence

This Calendar function was designed to return True if the next bar is the Nth *Occurrence* of the weekday name. Otherwise this function will return False. For example, NextBarWeekDayOccurrence ("Tuesday", 2) returns true if the next bar is the second Tuesday of the month. The NextBarWeekDayOccurrence was only intended for use within mechanical systems, but since it returns True/False it can also be used for highlight bars.

Function:

NextBarWeekDayOccurrence (WeekDayName, Occurrence)

Inputs:

WeekDayName – This is the weekday name of the day that you are looking for. Please note that the weekday name must be enclosed by quotation marks (Please see example below).

Occurrence – This is the number occurrence of the weekday you are looking for. If this input is negative, the function will look from the end of the current month back. In the example below we are looking for the last Friday of the month., so we are using "-1" as the *Occurrence*.

Example:

In this example, we want to create a highlight bar that highlights the last Friday of each month.

NextBarWeekDayOccurrence ("Friday", -1)

NextBarYear

This Calendar function was designed to return the year portion of the date for the next bar. For example: If the next bar is in the year of 1999, then this NextBarYear function will return 99. This function, like many of the other "Next Bar" functions, was only intended to be used within mechanical system rules. This NextBarYear function provides an easier way to specify conditions that trigger trades based on specific calendar conditions. Since this function returns a two digit year value for the next bar, it can also be forced to return True/False, by adding a math operator (<, >, =) followed by the specific year that the next bar must fall in, in order to return a True value.

Function:

NextBarYear

Inputs:

None

Example:

In this example, the rule that we would like to use is only applicable in the year 2000 or after.

IF NextBarYear >= 00

November

This Calendar function was designed to return True if the current bar is in the month of November. Otherwise, this function will return False. This function was intended to be used in mechanical system rules, Filter Criteria/Scans, as well as some custom functions.

Function:

November

Inputs:

None

Example:

In this example, we would like *IF November = False*
the current bar to be in any other month besides November in order to trigger a
trade for the next bar.

Occurrences

This Misc function was designed to return the number of *Conditions* that have happened over a specified amount of time (*The Last N Bars*). The Occurrence function was intended for use within mechanical system rules, Filter Criteria/Scans, and/or custom functions. It is most commonly used in the condition, or formula, within rules, or custom functions in order to specify the number of times that a *Condition* must have happened before an action takes place. Since this function returns a number (number of times the Condition took place), it can also be forced to return True/False values. Please see example below for an example of forcing a True/False return.

Function:

Occurrence (Condition, The Last N Bars)

Inputs:

Condition – This input is the actual pattern or market move that the *Occurrences* function locates, and then returns the number of times it has taken place within *The Last N Bars*.

The Last N Bars – This input is the total amount of bars that the *Occurrences* function will use to locate the number of times the *Condition* has taken place. If this input is set to “0”, the function will use the full history of the data file.

Example:

In this example, we would like *Occurrences (Close < Close.1, 50) >= 15*
a Criteria that filters for all securities that have had 15 or more down closes within the last 50 bars
of data.

October

This Calendar function was designed to return True if th e current bar is in the month of October. Otherwise, this function will return False. This function was intended to be used in mechanical system rules, Filter Criteria/Scans, as well as some custom functions.

Function:

October

Inputs:

None

Example:

In this example, we would like *IF October and ADX (7) > 60*
to enter a position in a market only if the current bar is in October, and the most recent 7 bar ADX value is greater than 60.

OutsideBar

This Indicator function was designed to return True if the current, most recent bar satisfies the conditions for an OutsideBar. The criteria for an OutsideBar is a bar that traded both higher and lower than the previous bar (higher high and lower low). This function can be used in mechanical system rules, Filter Criteria/Scans, and/or charting indicators/custom functions. Its most common use is either in a highlight bar (custom function), or in system rules, triggering a trade signal.

Function:

OutsideBar

Inputs:

None

Example:

In this example, we would like *OutsideBar = True and Close > Close.1*
to highlight all bars that are outside bars, as well as having a closing price that is greater than the previous bars closing price.

Power

This Math function was designed to return the result of raising the *Number* input by the *PowerValu*e input. Input 1 (*Number)* is raised by the power of Input 2 (*PowerValue)*.

Function:

Power (Number, PowerValue)

Inputs:

Number – This is the number (or math operation that returns a number value) to be raised by the *PowerValu*e. This input requires a number or a formula that produces a number.

PowerValue – This input is the power that Input 1 (*Number)* is raised by. Both inputs require a number or a number variable.

Example:

In this example, we are raising the current bars closing price by a power of 3.

Power (Close, 3)

PriceToDollars

This Math function was designed to convert a price movement of the current market symbol to a dollar amount. This function will return the dollar value for the specified *Price* move. It is most commonly used within mechanical system rules, specifying the fill price for certain trade signals.

Function:

PriceToDollars (Price)

Inputs:

Price – This input is the price movement to be converted to a dollar amount.

Example:

In this example of an action or order for next bar, we want to Exit at our Entry Price plus 3 price moves.

IF True
THEN Long Exit
At Entry Price + PriceToDollars (3)

RelativeStrengthRatio

This Indicator function is also known by the name Price Ratio, and was designed to compare the performance of one security relative to another. It is calculated by dividing the close of the first security by the close of the second and returns the difference as a number value. It is most commonly used within mechanical system rules, Filter Criteria/Scans, and/or charting indicators/custom functions.

Function:

RelativeStrengthRatio (PrimaryValue, BaseValue)

Inputs:

PrimaryValue – This input is the closing price of the primary market.

BaseValue – This input is the closing price of the market that the primary is being compared to.

Example:

In this example we would like *RelativeStrengthRatio (Gold, Bonds)*
a custom indicator (returning a value) that plots the ratio between Gold and Bonds.

Remainder

This Math function was designed to return the remainder after dividing two numbers. For example, 9 divided by 2, is equal to 4 with a remainder of 1. Therefore this function will return only the 1. This function was intended for use within mechanical system rules, as well as Filter Criteria/Scans.

Function:

Remainder (NumberToDivide, DivideByThisNumber)

Inputs:

NumberToDivide – This input is the numerator in the equation. In the equation 9/2, the 9 is your *NumberToDivide.*

DivideByThisNumber – This input is the denominator in the equation. Divide the *NumberToDivide* by this input. In the equation, 9/2, the 2 is your *DivideByThisNumber* input.

Example:

In this example we would Remainder (Close, Close.1)*100
like to multiply the remainder of the close divided by the close yesterday, by 100.

Round

This Math function was designed to round a number (as a decimal) to either a whole number or a specified decimal place. For example, *Round (35.748)* will return 36, but *Round (35.748, 2)* will return 35.75.

Function:

Round (Expression, NumDigits)

Inputs:

Expression – This input is the number value, or function value to be rounded to a certain *NumDigits*.

NumDigits – This input is used to specify how many decimal places after the decimal to round to. For example, Round (35.748, 2). This will return 35.75, because we specified 2 decimal places after the decimal.

Example:

In this example we would like *Round (Close.1 - Close)*
to round the difference between the close yesterday and the close today.

Seasonal

This Indicator function was designed to be used internally in our Seasonal indicators such as, Seasonal Trend. This internal function provides a way to specify certain values (see inputs below) that the Seasonal Trend indicator needs in order to calculate.

Function:

Seasonal (PriceMode, Method, LookBack, StdDev)

Inputs:

Price Mode – This input is to designate the certain price, or price calculation to be used in the *Seasonal* function. The following is a list of *Price Mod*es that are available:

0 = Open	4 = Average of High and Low
1 = High	5 = Average of Open and Close
2 = Low	6 = Average of High, Low and Close
3 = Close	7 = Average of Open, High, Low and Close

Method – This input allows custom specification for the type of average used in the *Seasonal* functions calculation. Following is a list of averages that are available:

1 – Simple
2 – Exponential
3 – Weighted
4 – Modified Exponential

LookBack – This input is the number of years that the *Seasonal* function will *LookBack* and use for calculation.

StdDev – This input is the standard deviation amount above or below the average that is used in the *Seasonal* function.

Example:

In this example we are creating a seasonal indicator using a simple average of the high and low.

Seasonal (4, 1, 3, 0)

SecurityType

This Price function was designed to return the type of security, such as “F” for Futures, “S” for Stocks, “I” for Indexes, or “M” for Mutual Funds. This function was only intended for use with mechanical system rules, as well as Filter Criteria/Scans.

Function:

SecurityType

Inputs:

None

Example:

In this example we would like to add (as a condition) a criteria that will filter for just futures. *Remember that this is just an example, there are other ways to accomplish this within a criteria.

SecurityType = F

September

This Calendar function was designed to return True if the current price bar is in the month of September. Otherwise this function will return False. This function was intended for use within mechanical system rules, Filter Criteria/Scans, and/or custom functions.

Function:

September

Inputs:

None

Example:

In this example, we would like to enter a position in a market only if the current bar is in September, and the most recent 7 bar ADX value is greater than 60.

IF September and ADX (7) > 60
THEN Enter Long
At Market

SessionEndTime

This Calendar function was designed to return the market ending time for the current day in a HHMM format (Hour Hour Minute Minute). This Calendar function was only intended for use within mechanical system rules, but can be used in highlight bar True/False formulas.

Function:

SessionEndTime

Inputs:

None

Example:

In this example we are creating a highlight bar. If the session end time is greater than 2:00 p.m., then highlight the first bar of the day, otherwise return False.

IFF (Session End Time > 1400 , FirstBarOfDay (0) = True , False) = True

SessionStartTime

This Calendar function was designed to return the market start time for the current day in a HHMM format (Hour Hour Minute Minute). This Calendar function was only intended for use within mechanical system rules, but can be used in highlight bar True/False formulas.

Function:

SessionStartTime

Inputs:

None

Example:

In this example we are creating a highlight bar. If the session start time is greater than 8:00 p.m., then highlight the first bar of the day, otherwise highlight the last bar of the day.

IFF (Session Start Time > 0800 , FirstBarOfDay (0) = True , LastBarOfDay (0) = True) = True

---

ShiftBars

This Reserved/Trade function was designed to shift the bars backward or forward by a specified number of bars (*Number to Shift*) from the *Expression's* value. This function is most commonly used within custom functions, such as highlight bars and indicators.

Function:

ShiftBars (Expression, Number to Shift)

Inputs:

Expression – This input is the number, indicator, or custom function value that gets shifted by the *Number to Shift.*

Number to Shift – This input is the actual number of bars to shift the *Expressions* value. Positive values for this input will shift the bars forward, and negative values will shift the bars backwards.

Example:

In this example we would like to plot the 7 bar simple moving average shifted backward by 2 bars.

ShiftBars (MovingAvg (Close, 7), -2)

Sign

This Math function was designed to return the sign of a number. For example, it will return –1 for negative numbers, 0 for zero, and 1 for all positive numbers.

Function:

Sign (Expression)

Inputs:

Expression – This input is the number, indicator, custom function, and/or math equation that the Sign function will use to return the sign of the value of this input.

Example:

In this example we would like the sign function to return the sign of the close subtracted from the close 2 bars ago, multiplied by 100.

*Sign ((Close – Close.2)*100)*

Sine

This Math function is simply the Sine operation, returning the Sine of a *Number*.

Function:

Sine (Number)

Inputs:

Number – This input is the number that will be returned as the *Sine* of the number.

Example:

In this example we would like the sine of the open subtracted from the close multiplied by 100.

*Sine ((Open – Close)*100)*

SP500

This Data function was designed to return the closing price (of current bar) for the S&P 500 future contract.

Function:

SP500

Inputs:

None

Example:

In this example we would like the S&P 500 to have closed greater than the close of Gold as the condition or entry signal in a system rule.

IF SP500 > Gold

THEN Long Entry

At Market

Spread

This Indicator function was designed to compare two values (Primary Value, Base Value) and return the difference (as a number) between them. The formula simply subtracts the two values and returns the difference. It is most commonly used within mechanical system rules, Filter Criteria/Scans, and/or custom functions.

Function:

Spread (Primary Value, Base Value)

Inputs:

Primary Value – This input is the first value (indicator, price, etc.) that is to be compared to the *Base Value* in the *Spread* calculation.

Base Value – This input is the second value (indicator, price, etc.) that is to be used in the *Spread* calculation.

Example:

In this example we would like to create a Spread line based on the difference between the closes and opens.

Spread (Close, Open)

SqrRoot

This Math function was designed to return the square root of the expression specified by the user. It is most commonly used within custom functions, such as functions that return number values.

Function:

SqrRoot (Expression)

Inputs:

Expression – This input is the actual number, or indicator value that is used for the SqrRoot function. The SqrRoot function will return the square root of this input.

Example:

In this example we would like to take the square root of the most recent 7 bar ADX value and multiply the result by 100.

*SqrRoot (ADX (7))*100*

Square

This Math function was created to make squaring a number easier than writing a long formula. It simply squares the *Number* specified by the user and returns the result. The *Number* is simply multiplied by itself.

Function:

Square (Number)

Inputs:

Number – This input is the value that is squared. This input can be a number, or any other function that returns a number value.

Example:

In this example, we would like to square the closing price of yesterday.

Square (Close.1)

StdDevsAsPercentBelow

This Math function was designed to return the percentage below a specified number of standard deviations (*NumStdDevs*) from the average. A percentage returned that is less than 50 is for negative standard deviations, and above 50 is positive standard deviations. This most commonly used within other functions, such as standard deviation indicators.

Function:

StdDevsAsPercentBelow (NumStdDevs)

Inputs:

NumStdDevs – This input is the actual number of standard deviations away from the average that will be returned as a percentage..

Example:

In this example we would like the percentage returned, when a standard deviation of 2 is used. *StdDevsAsPercentBelow (2)*

StdDevsForPercent

This Math function was designed to return the number of standard deviations away from the average for the specified percentage (0 – 100). It will return numbers above 50 for positive percentage, and below 50 for negative percentage. This function is most commonly used within other functions, such as, functions that are indicators.

Function:

StdDevsForPercent (Percent)

Inputs:

Percent – This input is the percentage used to find the number of standard deviations away from the average in the calculation of the *StdDevsForPercent* function.

Example:

In this example we would like to use a 50 percent deviation. *StdDevsForPercent (50)*

StdDevsForValue

This Math function was designed to return the number of standard deviations that the *Value* is from the average of the *Expression* (for a specified number of bars). This is very similar to Microsofts Excel's "Standardize" function. It will return a negative number if the value is below the average.

Function:

StdDevsForValue (Expression, NumBars, Value)

Inputs:

Expression – This input is the price function, indicator function, or number averaged in the calculation of the *StdDevsForValue*.

NumBars – This input is the number of bars that the *StdDevsForValue* function uses to average the *Expression*.

Value – This input is the price function, or indicator function that the *StdDevsForValue* returns the number of standard deviations away from the *Expressions* average.

Example:

In this example we would S*tdDevsForValue (MovingAvg (Close, 18), 3, Close)* <= 2
like for the close to be deviated off of the 18 bar moving average by either 2, or less.

StopAndReverse

This Misc function was designed to allow the creation of generic stop and reverse type indicators. It is very similar to Wells Wilder's Parabolic indicator as well as the Volatility indicator. It will calculate a stop value below the prices in an uptrend (during a long position), and when the prices fall below the Long Stop value the direction is reversed and the "Stop" value is then calculated above the prices (for a short position).

Function:

StopAndReverse (LongStopOffset, FromHighestOf, ShortStopOffset, FromLowestOf, ReverseAtNextBarPenetration, Position, UseExtreme)

Inputs:

LongStopOffset – This input is the amount to be subtracted from the *FromHighest Of* value setting the stop value for a long position.

FromHighestOf – This input is the value that the *LongStopOffset* is subtracted from, setting the stop value for a long position. This input defaults to the most recent closing price.

ShortStopOffset – This input is the amount to be subtracted from the *FromLowestOf* value setting the stop value for a short position.

FromLowestOf – This input is the value that the *ShortStopOffset* is subtracted from, setting the stop value for a short position. This input defaults to the most recent closing price.

ReverseAtNextBarPenetration – This input designates when the reverse stop value is set. When this input is set to 0 or False, it will set the stop value (reversing the stop value) when the current bars close value crosses the last stop value. When a 1 or True is used, it will reverse the stop value when the next bar crosses the last stop value. This input defaults to 0 or False.

Position – This input designates which stop values are to be calculated. When this input is set to 0, the *StopAndReverse* function will return values for both long and short positions. If this input is set to 1, the *StopAndReverse* function will only return stop values for long positions. If this input is set to –1, the *StopAndReverse* function will only return stop values for short position. This input is defaulted to 0, displaying stop values for both long and short positions.

UseExtreme –

<u>Example:</u>

In this example, we would like to create a volatility indicator using the average true range multiplied 3, as the stop offset. *Notice that only the stop offset had to be set, the rest of the inputs are left at their default values.

StopAndReverse (AvgTrueRange (7) 3)*

<u>StringCompare</u>

This Misc function was designed to allow the comparison of two values (Text 1, Text 2). If the values are equal then the function will return True. Otherwise, this function will return False.

<u>Function:</u>

StringCompare (Text 1, Text 2)

<u>Inputs:</u>

Text 1 – This input is the first of the two comparison values. The *StringCompare* function will return True if this input equals the *Text 2* input.

Text 2 – This input is the second of the two comparison values. The *StringCompare* function will return True if this input equals the first input, *Text 1*.

<u>Example:</u>

In this example, we would like to compare the close with the open values and have a highlight bar highlight bars when the close equals the open.

StringCompare (Close, Open)

StringContains

This Misc function was designed to identify whether *String 1* is contained in *String 2*. If *String 1* was found in *String 2*, then this function will return True. Otherwise this function will return False.

Function:

StringContains (String 1, String 2)

Inputs:

String 1 – This input is the string of numbers or indicator values that the *StringContains* function searches for within the value of *String 2*.

String 2 – This input is the string of numbers or indicator values that the *StringContains* function searches in to find the value of *String 1*.

Example:

In this example we would like to know whether the open, high, or close values are found within the 18 bar moving average values 2 bars ago, 1 bar ago, or on the current bar.

StringCompare (Open or High or Low or Close,
MovingAvg (Close, 18).2 or low
MovingAvg (Close, 18).1
or MovingAvg (Close, 18))

SwingHigh

This Misc function was designed to return the price of the specified *Occurrence* of a Swing High. It uses four inputs designating, the number of bars used, the *Method* used, the type of *AltMode* used, and the specific *Occurrence* of the High Swing that it is looking for. This SwingHigh function was intended for use within mechanical system rules, as well as Filter Criteria/Scans. It is most commonly used when referencing the price of a certain occurrence of a SwingHigh.

Function:

SwingHigh (Strength, Method, AltMode, Occurrence)

Inputs:

Strength – This input is the number of bars that the *SwingHigh* function searches within in order to find the specific *Occurrence* of a SwingHigh.

Method – This input allows for the specification of the type of swings that the SwingHigh function looks for. Following is a list of available input values that can be used.

0 – Normal method that is used when the number of bars are higher/lower on both sides.
1 – Used when a bar closes below the low of the high day.
2 – Used when a bar closes below specified “X” number of consecutive closes.
3 – Used when a bar is completely below the low of the high day.
4 – Used to find any of #1-#3 conditions above.
5 – Used to find either #2 or #3.

AltMode – This input allows users to specify the order in which swing points are calculated.

0 – This will not alternate highs or lows, showing all possible swing points.
1 – This inserts alternates between two like swings. IE. If there are two high swing points in a row, this AltMode inserts a low swing point at the lowest low between them.
2 – This will force the confirmation of opposite swings. IE If there is a high swing point, this AltMode will ignore any additional high swings until after the next low swing

Occurrence – This input is the specific "Happening" of a high swing that the SwingHigh function is to return the value/price for.

Example:

In this example we would *High > Highest (Swing High (3, 0, 1, 1), 60).1*
like the current bars high to be greater than the highest most recent swing high value in the last 60 bars.

SwingHighBar

This Misc function was designed to return the number of bars since the specified *Occurrence* of a *SwingHighBar*. It uses four inputs designating, the number of bars used, the *Method* used, the type of *AltMode* used, and the specific *Occurrence* of the High Swing that it is looking for. This SwingHighBar function was intended for use within mechanical system rules, as well as Filter Criteria/Scans. It is most commonly used when referencing the bars since a certain occurrence of a SwingHigh.

Function:

SwingHighBar (Strength, Method, AltMode, Occurrence)

Inputs:

Strength – This input is the number of bars that the *SwingHighBar* function searches within in order to find the specific *Occurrence* of a high swing.

Method – This input allows for the specification of the type of swings that the *SwingHighBar* function looks for. Following is a list of available input values that can be used.

0 – Normal method that is used when the number of bars are higher/lower on both sides.
1 – Used when a bar closes below the low of the high day.
2 – Used when a bar closes below specified "X" number of consecutive closes.
3 – Used when a bar is completely below the low of the high day.
4 – Used to find any of #1-#3 conditions above.
5 – Used to find either #2 or #3.

AltMode – This input allows users to specify the order in which swing points are calculated.

0 – This will not alternate highs or lows, showing all possible swing points.
1 – This inserts alternates between two like swings. IE. If there are two high swing points in a row, this AltMode inserts a low swing point at the lowest low between them.
2 – This will force the confirmation of opposite swings. IE If there is a high swing point, this AltMode will ignore any additional high swings until after the next low swing

Occurrence – This input is the specific "Happening" of a high swing that the *SwingHighBar* function is to return the value/price for.

Example:

In this example we would *BarsSince (ADX (7) < 60, 1, 60) < SwingHighBar (60, 1, 0, 1)* like the bars since the last time the 7 bar ADX value was less than 60, to be less than the bars since the last (most recent) swing high bar.

SwingHighBarCustom

This Misc function was designed to return the number of bars since the specified occurrence of a "Custom" swing high bar, using a customized (user specified) *Expression* to calculate the Swing Points. It will also allow for Strength, AltMode, and Occurrence inputs. This function was only intended to be used within mechanical system rules, Filter Criteria/Scans, and/or other custom functions.

Function:

SwingHighBarCustom (Expression, Strength, AltMode, Occurrence)

Inputs:

Expression – This input is the price function (or value) that the *SwingHighBarCustom* function uses to calculate the swing points. It uses a default of the closing prices. This input is what makes this a "Custom" function, allowing the ability to specify the price or indicator function that the function uses.

Strength – This input is the number of bars that the *SwingHighBarCustom* function searches within in order to find the specific *Occurrence* of a high swing.

AltMode – This input allows users to specify the order in which swing points are calculated.

0 – This will not alternate highs or lows, showing all possible swing points.
1 – This inserts alternates between two like swings. IE. If there are two high swing points in a row, this AltMode inserts a low swing point at the lowest low between them.
2 – This will force the confirmation of opposite swings. IE If there is a high swing point, this AltMode will ignore any additional high swings until after the next low swing

Occurrence – This input is the specific "Happening" of a high swing that the *SwingHighBarCustom* function is to return the value/price for.

Example:

In this example we would like *SwingHighBarCustom (StochK (14,3), 20, 0, 1) < 3* the bars since the last (most recent) swing high bar of the StochK, to be less than 3 in the last 20 bars.

SwingHighCustom

This Misc function was designed to return the price of the specified *Occurrence* of a "Custom" swing high. It uses five inputs designating the price or indicator function used, the number of bars used, the *Method* used, the type of *AltMode* used, and the specific *Occurrence* of the High Swing that it is looking for. This SwingHighCustom function was intended for use within mechanical system rules, as well as Filter Criteria/Scans. It is most commonly used when referencing the price of a certain occurrence of a SwingHigh.

Function:

Swing High Custom (Expression, Strength, AltMode, Occurrence)

Inputs:

Expression – This input is the price function (or value) that the *SwingHighCustom* function uses to calculate the swing points. It uses a default of the closing prices. This input is what makes this a "Custom" function, allowing the ability to specify the price function (or value) that the function uses.

Strength – This input is the number of bars that the *SwingHighCustom* function searches within in order to find the specific *Occurrence* of a high swing.

AltMode – This input allows users to specify the order in which swing points are calculated.

0 – This will not alternate highs or lows, showing all possible swing points.
1 – This inserts alternates between two like swings. IE. If there are two high swing points in a row, this AltMode inserts a low swing point at the lowest low between them.
2 – This will force the confirmation of opposite swings. IE If there is a high swing point, this AltMode will ignore any additional high swings until after the next low swing

Occurrence – This input is the specific "Happening" of a high swing that the *SwingHighCustom* function is to return the value/price for.

<u>Example:</u>

StochK (14,3) > SwingHighCustom (StochK (14,3), 20, 0, 1)

In this example we would like the current StochK value to be greater than the last swing high value of the StochK in the last 20 bars of data.

<u>SwingLow</u>

This Misc function was designed to return the actual price of the specified Occurrence of a Swing Low point. It uses four inputs designating the number of bars used, the *Method* used, the type of *AltMode* used, and the specific *Occurrence* of the low swing that it is looking for. This SwingLow function was intended for use within mechanical system rules, as well as Filter Criteria/Scans. It is most commonly used when referencing the price of a certain occurrence of a SwingLow.

<u>Function:</u>

SwingLow (Strength, Method, AltMode, Occurrence)

<u>Inputs:</u>

Strength – This input is the number of bars that the *SwingHigh* function searches within in order to find the specific *Occurrence* of a SwingHigh.

Method – This input allows for the specification of the type of swings that the SwingHigh function looks for. Following is a list of available input values that can be used.

0 – Normal method that is used when the number of bars are higher/lower on both sides.
1 – Used when a bar closes below the low of the high day.
2 – Used when a bar closes below specified "X" number of consecutive closes.
3 – Used when a bar is completely below the low of the high day.

4 – Used to find any of #1-#3 conditions above.
5 – Used to find either #2 or #3.

AltMode – This input allows users to specify the order in which swing points are calculated.

0 – This will not alternate highs or lows, showing all possible swing points.
1 – This inserts alternates between two like swings. IE. If there are two high swing points in a row, this AltMode inserts a low swing point at the lowest low between them.
2 – This will force the confirmation of opposite swings. IE If there is a high swing point, this AltMode will ignore any additional high swings until after the next low swing

Occurrence – This input is the specific "Happening" of a high swing that the SwingHigh function is to return the value/price for.

Example:

In this example we would like the current bars low to be less than the lowest most recent swing low value in the last 60 bars.

Low > Lowest (Swing Low (3, 0, 1, 1), 60).1

SwingLowBar

This function was designed to return the number of bars since the specified occurrence of a Swing Low Bar. It uses four inputs designating the number of bars used, the *Method* used, the type of *AltMode* used, and the specific *Occurrence* of the High Swing that it is looking for. This *SwingLowBar* function was intended for use within mechanical system rules, as well as Filter Criteria/Scans. It is most commonly used when referencing the bars since a certain occurrence of a SwingLow.

Function:

SwingLowBar (Strength, Method, AltMode, Occurrence)

Inputs:

Strength – This input is the number of bars that the *SwingHighBar* function searches within in order to find the specific *Occurrence* of a high swing.

Method – This input allows for the specification of the type of swings that the *SwingHighBar* function looks for. Following is a list of available input values that can be used.

0 – Normal method that is used when the number of bars are higher/lower on both sides.
1 – Used when a bar closes below the low of the high day.
2 – Used when a bar closes below specified "X" number of consecutive closes.
3 – Used when a bar is completely below the low of the high day.
4 – Used to find any of #1-#3 conditions above.
5 – Used to find either #2 or #3.

AltMode – This input allows users to specify the order in which swing points are calculated.

0 – This will not alternate highs or lows, showing all possible swing points.
1 – This inserts alternates between two like swings. IE. If there are two high swing points in a row, this AltMode inserts a low swing point at the lowest low between them.
2 – This will force the confirmation of opposite swings. IE If there is a high swing point, this AltMode will ignore any additional high swings until after the next low swing

Occurrence – This input is the specific "Happening" of a high swing that the *SwingHighBar* function is to return the value/price for.

Example:

In this example we would *BarsSince (ADX (7) > 60, 1, 60) < SwingLowBar (60, 1, 0, 1)* like the bars since the last time the 7 bar ADX value was above 60, to be less than the bars since the last (most recent) swing low bar.

SwingLowBarCustom

This Misc function was designed to return the number of bars since the specified occurrence of a "Custom" swing low bar using a customized (user specified) *Expression* to calculate the Swing Points. It will also allow for Strength, AltMode, and Occurrence inputs. This function was only intended to be used within mechanical system rules, Filter Criteria/Scans, and/or other custom functions.
Function:

SwingLowBarCustom (Expression, Strength, AltMode, Occurrence)

Inputs:

Expression – This input is the price or indicator function that the *SwingLowBarCustom* function uses to calculate the swing points. It uses a default of the closing prices. This input is what makes this a "Custom" function, allowing the ability to specify the price function that the function uses.

Strength – This input is the number of bars that the *SwingLowBarCustom* function searches within, in order to find the specific *Occurrence* of a high swing.

AltMode – This input allows users to specify the order in which swing points are calculated.

0 – This will not alternate highs or lows, showing all possible swing points.

1 – This inserts alternates between two like swings. IE. If there are two high swing points in a row, this AltMode inserts a low swing point at the lowest low between them.
2 – This will force the confirmation of opposite swings. IE If there is a high swing point, this AltMode will ignore any additional high swings until after the next low swing

Occurrence – This input is the specific "Happening" of a low swing that the *SwingLowBarCustom* function is to return the value/price for.

Example:

In this example we would like *SwingLowBarCustom (StochK (14,3), 20, 0, 1) < 3* the bars since the last (most recent) swing low bar of the StochK, to be less than 3 in the last 20 bars.

SwingLowCustom

This Misc function was designed to return the price of the specified *Occurrence* of a "Custom" swing low. It uses five inputs designating the price or indicator function used, the number of bars used, the *Method* used, the type of *AltMode* used, and the specific *Occurrence* of the low swing that it is looking for. This SwingLowCustom function was intended for use within mechanical system rules, as well as Filter Criteria/Scans. It is most commonly used when referencing the price of a certain occurrence of a SwingLow.

Function:

SwingLowCustom (Expression, Strength, AltMode, Occurrence)

Inputs:

Expression – This input is the price function (or value) that the *SwingLowCustom* function uses to calculate the swing points. It uses a default of the closing prices. This input is what makes this a "Custom" function, allowing the ability to specify the price function (or value) that the function uses.

Strength – This input is the number of bars that the *SwingLowCustom* function searches within in order to find the specific *Occurrence* of a low swing.

AltMode – This input allows users to specify the order in which swing points are calculated.

0 – This will not alternate highs or lows, showing all possible swing points.

1 – This inserts alternates between two like swings. IE. If there are two high swing points in a row, this AltMode inserts a low swing point at the lowest low between them.
2 – This will force the confirmation of opposite swings. IE If there is a high swing point, this AltMode will ignore any additional high swings until after the next low swing

Occurrence – This input is the specific "Happening" of a low swing that the *SwingLowCustom* function is to return the value/price for.

Example:

In this example we would like the current StochK value to be lessr than the last swing low value of the StochK in the last 20 bars of data.

StochK (14,3) < SwingLowCustom (StochK (14,3), 20, 0, 1)

SwingPointsCustom

This Misc function was designed to chart Swing Points allowing the user to attach the SwingPointsCustom to another price or indicator function, such as the open, high, low, or close. It also allows the *Expression* to be expressed as an indicator, such as ADX, or a Moving Average. This function is most commonly used when creating custom swing point functions to be plotted on a chart.

Function:

SwingPointsCustom (Expression, Strength, AltMode)

Inputs:

Expression – This input is the price function (or value) that the *SwingPointsCustom* function uses to calculate the swing points. It uses a default of the closing prices. This input is what makes this function a "Custom" function, allowing the ability to specify the price (or value) that the SwingPoints use.

Strength – This input is the number of bars that the *SwingPointsCustom* function uses.

AltMode – This input allows users to specify the order in which the swing points are calculated.

0 – This will not alternate highs or lows, showing all possible swing points.
1 – This inserts alternates between two like swings. IE. If there are two high swing points in a row, this AltMode inserts a low swing point at the lowest low between them.
2 – This will force the confirmation of opposite swings. IE If there is a high swing point, this AltMode will ignore any additional high swings until after the next low swing

Example:

In this example we would like to create a swing points indicator that plots all possible swing points on the Ultimate Oscillator indicator (instead of swing points attached to the price).

SwingPointsCustom (UltimateOsc (7, 14, 28), 6, 0)

Thursday

This Calendar function was designed to return True if the current price bar is a Thursday. Otherwise this function will return False. This function was intended for use within mechanical system rules, Filter Criteria/Scans, and/or custom functions.

Function:

Thursday

Inputs:

None

Example:

In this example we would like *Thursday and Close > MovingAvg (Close, 18)* to highlight all daily bars that are Thursdays and the current bars close closed above the 18 bar moving average.

TickMove

This Price function was designed to return the actual tick move of a security. For example, for T-Bonds this value is .03125. This function was intended for use within mechanical system rules, as well as custom functions.

Function:

TickMove

Inputs:

None

Example:

In this example we need *IF True* to exit a long position at a price of our Entry price *THEN Long Exit* plus 3 tick moves.

*At Entry Price + TickMove * 3*

TickValue

This Price function was designed to return the actual dollar value of a TickMove for a given security. For example, for T-Bonds this value is $31.25 for 1 TickMove (.03125). This function was intended for use within mechanical system rules, as well as custom functions.

Function:

TickValue

Inputs:

None

Example:

In this example we need *IF True* to exit a long position at a price of our Entry price *THEN Long Exit* plus a tick value.

At Entry Price + TickValue

TickVolume

This Price function was designed to return the number of ticks or trades in a market for the current bar. It is calculated by adding the TicksUp to the TicksDown. To receive these values you must be a subscriber to Genesis Intraday data. This function was only intended for use within mechanical system rules.
Function:

TickVolume

Inputs:

None

Example:

In this example we would like the tick volume today to be greater than it was 2 days ago.

IF TickVolume > TickVolume.2

TicksDown

This Price function was designed to return the number of ticks or trades that traded lower than the last trade at a different price. To receive these values you must be a subscriber to Genesis Intraday data. This function was intended for use within mechanical system rules, Filter Criteria/Scans, and/or custom functions.

Function:

TicksDown

Inputs:

None

Example:

In this example we would like the current amount of down ticks to be less than the down ticks of the first bar of yesterday's data.

TicksDown. FirstBarOfDay (0) < TicksDown. FirstBarOfDay (1)

TicksUp

This Price function was designed to return the number of ticks or trades that traded higher than the last trade at a different price. To receive these values you must be a subscriber to Genesis Intraday data. This function was intended for use within mechanical system rules, Filter Criteria/Scans, and/or custom functions.

Function:

TicksUp

Inputs:

None

Example:

In this example we would like the current amount of up ticks to be more than the up ticks of the first bar of yesterday's data.

TicksUp. FirstBarOfDay (0) < TicksUp. FirstBarOfDay (1)

Time

This Calendar function was designed to return the time of the current bar as a number in the format of "HHMM" (Hour Hour Minute Minute). For example, 930 for 9:30 a.m., 1615 for 4:15 p.m. This function will also return True/False for a highlight bar if it is written correctly (e.g. Time = 930).

Function:

Time

Inputs:

None

Example:

In this example, we would like to highlight all of the 9:30 a.m bars.

Time = 0930

TimeToMinutes

This Calendar function was designed to convert "Time" (HHMM) to minutes since midnight. This function may not necessarily be used by the common user, but for some, it may be useful. It was only intended for use within mechanical system rules, and some Filter Criteria/Scans.

Function:

TimeToMinutes (HHMM)

Inputs:

HHMM – This input is the actual time (expressed in 24 hour format) to be converted to the amount of minutes since midnight. For example, 930 (HHMM) = 570 (Minutes since midnight).

TradedToday

This Indicator function was designed to return True if a trade has already been entered on the same day as the next bar. Otherwise this function will return False. This function was intended for use within mechanical system rules only.

Function:

TradedToday

Inputs:

None

Example:

In this example we would like of our condition to state that there has not been a trade filled

IF NOT TradedToday part

OR today.

IF TradedToday = False

TradingDayOfMonth

This Calendar function was designed to return the trading day of the month as a number for the current bar. For example if today is 03/03/03, then this function will return 1, because the 3rd of March of 2003 is the very first trading day of the month (and it is not a holiday). This function was intended for use within mechanical system rules, as well as Filter Criteria/Scans and custom functions. You will see in the example below that we have "=1" after the function. This forces the confirmation (True/False) of our condition. If the TradingDayOfMonth does not equal 1, then our condition is false, canceling any trade signal.

Function:

TradingDayOfMonth

Inputs:

None

Example:

In this example we would like *IF TradingDayOfMonth = 1*
part of our condition to state that the current bar is the first trading
THEN Exit Long day of the month.
At Market

TradingDayOfWeek

This Calendar function was designed to return the trading day of the week as a number for the current bar. For example if today is Monday (and it is not a holiday), this function will return 1. This function was intended for use within mechanical system rules, as well as Filter Criteria/Scans and custom functions. You will see in the example below that we have "=1" after the function. This forces the confirmation (True/False) of our condition. If the TradingDayOfWeek does not equal 1, then our condition is false, canceling any trade signal.

Function:

TradingDayOfWeek

Inputs:

None

Example:

In this example we would like *IF TradingDayOfWeek = 1*
part of our condition to state that the current bar is the first trading
THEN Exit Long day of the week.
At Market

TradingDayOfYear

This Calendar function was designed to return the trading day of the year as a number for the current bar. For example if today is 05/30/03 (and it is not a holiday), this function will return 103. This function was intended for use within mechanical system rules, as well as Filter Criteria/Scans and custom functions. You will see in the example below that we have ">=103" after the function. This forces the confirmation (True/False) of our condition. If the TradingDayOfYear is not greater than or equal to 103 trading day of the year, then our condition is false, canceling any trade signal.

Function:

TradingDayOfYear

Inputs:

None

Example:

In this example we would like part of our condition to state that the current bar is greater than or equal to the 103rd trading day of the year.

IF TradingDayOfYear >= 103
THEN Enter Long
At Market

TradingDaysAfterHoliday

This Calendar function was designed to return the number of trading days since the last holiday (as a number) to the current bar. This function was intended for use within mechanical system rules, as well as Filter Criteria/Scans and custom functions. You will see in the example below that we have "=1" after the function. This forces the confirmation (True/False) of our condition. If the TradingDayAfterHoliday is not equal to 1, then our condition is false, canceling any trade signal for the next bar.

Function:

TradingDaysAfterHoliday

Inputs:

None

Example:

In this example we would like to place a trade on the second trading day after a holiday.

IF TradingDaysAfterHoliday = 1
THEN Enter Long
At Market

TradingDaysLeftBeforeHoliday

This Calendar function was designed to return the number of trading days from the current bar to the very next market holiday. This function was intended for use within mechanical system rules, as well as Filter Criteria/Scans and custom functions. You will see in the example below that we have "=2" after the function. This forces the confirmation (True/False) of our condition. If the TradingDayLeftBeforeHoliday is not equal to 2, then our condition is false, canceling any trade signal for the next bar.

Function:

TradingDaysLeftBeforeHoliday

Inputs:

None

Example:

In this example we would like place a trade on the day before a holiday.

IF TradingLeftBeforeHoliday = 2 to

THEN Exit Long

At Market

TradingDaysLeftInMonth

This function was designed to return the number of trading days left in the current month from the current bar forward. This function was intended for use within mechanical system rules, as well as Filter Criteria/Scans and custom functions. You will see in the example below that we have ">=10" after the function. This forces the confirmation (True/False) of our condition. If the TradingDaysLeftInMonth is not greater than or equal to 10, then our condition is false, canceling any trade signal for the next bar.

Function:

TradingDaysLeftInMonth

Inputs:

None

Example:

In this example we would like to place a trade if there are 10 trading days or more left in the month.

IF TradingDaysLeftInMonth >= 10

THEN Exit Long current

At Market

True

This Misc function was designed to always return True. It is most commonly used when a condition must be true. This function is one of the functions that was intended for use within mechanical system rules, Filter Criteria/Scans, and custom functions.

Function:

True

Inputs:

None

Example:

In this example we would like the previous bar to equal an outside bar, and the current close to be greater than the close 2 bars ago.

OutsideBar.1 = True And Close > Close.2

TrueInsideBar

This Indicator function was designed to return True if the current bar is in fact a bar that did not trade higher or lower than the preceding bars true high and true low, and whose range is less than or equal to the preceding bar. This function was intended for use within mechanical system rules, as well as Filter Criteria/Scans and custom functions.

Function:

TrueInsideBar

Inputs:

None

Example:

In this example we would like to enter a trade when the most recent True Inside Bar was less than or equal to 2 bars ago.

IF BarSince (TrueInsideBar, 1, 25) <= 2
THEN Long Entry
At Market

TrueOutsideBar

This Indicator function was designed to return True if the current bar is in fact a bar that traded both higher and lower than the preceding bars true high and true low. This function was intended for use within mechanical system rules, as well as Filter Criteria/Scans and custom functions.

Function:

TrueOutsideBar

Inputs:

None

Example:

In this example we would like to enter a trade when the most recent True Outside Bar was less than or equal to 5 bars ago.

IF BarSince (TrueOutsideBar, 1, 25) <= 5
THEN Long Entry
At Market

Tuesday

This Calendar function was designed to return True if the current price bar is a Tuesday. Otherwise this function will return False. This function was intended for use within mechanical system rules, Filter Criteria/Scans, and/or custom functions.

Function:

Tuesday

Inputs:

None

Example:

In this example we would like to highlight all bars that are tuesdays and are also inside bars.

Tuesday And InsideBar

UpRange

This Indicator function was designed to return True if a bar is an up range bar (higher high, and higher low than the previous bar). Otherwise this indicator function will return False. This function was intended for use within mechanical system rules, Filter Criteria/Scans, and/or custom functions.

Function:

UpRange

Inputs:

None

Example:

In this example we would like *Thursday And UpRange* to highlight all bars that are Thursdays, and are also UpRange bars.

VolatilityLongStop

This Indicator function was designed to set stop prices, according to Wells Wilders formula, when entered into a long position. It is calculated by using the Average True Range of 7 bars multiplied by constant (K). Please refer to Wells Wilders book "New Concepts in Technical Trading Systems", for more information on his Volatility indicators.

Function:

VolatilityLongStop (K)

Inputs:

K – This input is the constant value that the Average True Range is multiplied by in the Volatility Stop indicators. This value can be changed when using this VolatilityLongStop indicator, by changing the (K) value in the function, once it is referenced within a system rule, or when creating a custom function.

Example:

In this example we would like *VolatilityLongStop (5)* to create a custom Volatility long stop function that uses a (K) value of 5 rather than the standard (K) value of 3.

VolatilityShortStop

This Indicator function was designed to set stop prices, according to Wells Wilders formula, when entered into a short position. It is calculated by using the Average True Range of 7 bars multiplied by constant (K). Please refer to Wells Wilders book "New Concepts in Technical Trading Systems", for more information on his Volatility indicators.

Function:

VolatilityShortStop (K)

Inputs:

None

Example:

In this example we would like *VolatilityLongStop (5)* to create a custom Volatility short stop function that uses a (K) value of 7 rather than the standard (K) value of 3.

Wednesday

This Calendar function was designed to return True if the current price bar is a Wednesday. Otherwise this function will return False. This function was intended for use within mechanical system rules, as well as Filter Criteria/Scans and/or custom functions.

Function: *Wednesday*

Inputs:

None

Example:

In this example, we would like *Wednesday And InsideBar* to highlight all bars that are Wednesday and are also inside bars.

WeekDayOccurrence

This Calendar function was designed to return True if the current bar is the Nth (specified in the inputs) *Occurrence* of a weekday (also specified in the inputs) during the month. For example, WeekDayOccurrence (“Tuesday”, 2) will return True if the current bar is in fact the second Tuesday of the month. If the *Occurrence* input is set to a negative number (such as –1) then it will check from the end of the month, returning True (when using –1) if it is the last *WeekDayName* of the month.

Function:

WeekDayOccurrence (WeekDayName, Occurrence)

Inputs:

WeekDayName – This is the weekday name the WeekDayOccurrence function is searching for. *Please note that the weekday name must be enclosed by quotation marks (Please see example below).

Occurrence – This input is the specific occurrence of the weekday you are looking for. For example: If this input is set to “2”, then the *WeekDayOccurrence* function will look for the 2nd “*WeekDayName*” of the month. If this input is negative, the function will look from the end of the current month back. In the example below we are looking for the last Tuesday of the month., so we are using “-1” as the *Occurrence*.

Example:

In this example, we would like *WeekDayOccurrence (“Tuesday”, -1)* to highlight all bars that are the last Tuesday of each month.

XOR

This Misc function was designed to return True if exactly one of the two conditions is true. This function is used when only one of the two conditions desired needs to be True, but not both. It is intended for use within mechanical system rules, Filter Criteria/Scans, and/or custom functions.

Function:

XOR (Condition 1, Condition 2)

Inputs:

Condition 1 – This input is the first of two conditions, where one of the two must be true. This input can be any thing from, *ADX (7) > 60*, to *Open.1 > Open.10*. Remember that this XOR function is searching for the Condition to equal True, so these *Condition 1 & 2* inputs must return true or false.

Condition 2 - This input is the second of the two conditions, whereas one of the two must be true. This input can be any thing from, ADX (7) > 60, to Open.1 > Open.10. Remember that this *XOR* function is searching for Conditions to equal True, so the *Condition 1 & 2* inputs must return true or false.

Example:

In this example we are would like *XOR (ADX (7) >= 60, ADX (7) <= 20)* either the 7 bar ADX value to be greater than or equal to 60, or the 7 bar ADX value to be less than or equal to 20.

Year

This Calendar function was designed to return the numeric year as a four digit number value. For example, Year = 2003. It was only intended for use in either Criteria/Scans or mechanical system rules. You will see in the example below, that we have taken a number return function (*Year*) and turned it into a True/False conditional by adding the operator “=”, followed by the four digit year we wish the current bar to be in.

Function:

Year

Inputs:

None

Example:

In this example we would like the most recent bar to be in the year 1999.	*Year = 1999*