ITI - Complete Documentation of Zorro and Custom Functions for Algorithmic Trading

ZORRO - FUNCTION REFERENCE

Complete Documentation of Zorro Functions for Algorithmic Trading

Version: 2.0 - November 2025

Includes: Zorro native functions + options.c + helpers.c libraries

SUMMARY

1. CHAPTER 1: TIME SERIES ANALYSIS

Below is a summary table of the available functions:

Chapter	Category	Number of functions
1	Time Series Analysis	221
2	Markets & Trading	92
3	Math, Time & Data	94
4	Data Structures & Arrays	64
5	System Control	32
6	Custom Libraries	53
7	Input/Output	88
8	User Functions	15
TOTAL FUNCTIONS		**659**

Time-series analysis functions are the heart of technical analysis in Zorro. This chapter presents comprehensive documentation of all the functions available for analyzing, transforming, and interpreting market data.

MAIN CATEGORIES:

- 📊 Technical Indicators: MACD, RSI, Stochastic, ADX, Bollinger Bands, etc.
- 📈 Moving Averages: SMA, EMA, WMA, DEMA, TEMA, KAMA, HMA, etc.
- 🔄 Oscillators: AC, ADO, AO, CCI, CMO, ROC, Williams %R, etc.
- 🌊 Digital Filters: Butterworth, Gaussian, Highpass, Lowpass, Bandpass, etc.
- 🔍 Cyclic Analysis: Hilbert Transform, Spectrum Analysis, Dominant Period/Phase, etc.
- 📉 Supports/Resistances: Pivot Points, Fractals, Donchian Channels, etc.
- Pattern Recognition: 60+ Candlestick patterns, ZigZag, etc.
- 📊 Statistics: Correlation, Covariance, Standard Deviation, Variance, etc.

Each function is documented with:

- ✓ Detailed description and use cases
- ✓ Complete syntax and parameters
- ✓ Return values and their interpretation
- ✓ Practical code examples
- ✓ Usage notes and best practices
- ✓ Related functions and common problems

TOTAL FUNCTIONS: 216

1.1. 📊 TECHNICAL INDICATORS

1.1.1. ADX - Average Directional Movement Index

Description

Measures the strength of a trend regardless of direction, with values from 0 to 100. Values above 25 indicate the presence of a trend, and above 50 indicate a strong trend. Calculated from the moving average of the directional movement indicator, it is often used with +DI and -DI to identify the direction and strength of the trend. It does not indicate direction, only intensity.

Syntax

var ADX(vars Series);
// or with specific periodvar ADX(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ADX indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ADXvars Price = series(priceClose());var value = ADX(Price);var prevvalue = ADX(Price)[1];
// Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ADX", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.1.2. ADXR - Average Directional Movement Rating

Description

A smoothed version of the ADX that calculates the average between the current ADX value and the ADX value from N periods ago (typically 14). It produces smoother signals and is less sensitive to temporary spikes than the standard ADX. Useful for filtering out false signals in choppy markets while still being able to identify significant trends.

Syntax

var ADXR(vars Series);
// or with specific periodvar ADXR(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ADXR indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // 1-hour timeframe

// Calculate ADXRvars Price = series(priceClose());var value = ADXR(Price);var prevvalue = ADXR(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ADXR", value);

// Example trading condition
if(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.1.3. Alligator - Alligator 3-line Indicator

Description

A moving average system developed by Bill Williams consisting of three smoothed lines: Jaw (blue, slow), Teeth (red, medium), and Lips (green, fast). When the lines are intertwined, the market is in consolidation ("sleeping"); when they separate, it indicates a strong trend ("hunting"). Prices above the lines indicate an uptrend; below, a downtrend.

Syntax

var Alligator(vars Series);
// or with specific periodvar Alligator(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Alligator indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Alligatorvars Price = series(priceClose());var value = Alligator(Price);
var prevvalue = Alligator(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Alligator", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.1.4. Aroon - Aroon Indicator

Description

A system of two indicators (Aroon Up and Aroon Down) that measure the time since the last high/low in N periods (typically 25). Values from 0 to 100: A high Aroon Up indicates a strong uptrend, a high Aroon Down indicates a strong downtrend. Crossovers between the two lines signal potential trend changes, while parallel values indicate consolidation.

Syntax

var Aroon(vars Series);
// or with specific periodvar Aroon(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Aroon indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Aroonvars Price = series(priceClose());var value = Aroon(Price);var prevvalue = Aroon(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Aroon", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.1.5. BBands - Bollinger Bands

Description

A three-line system: a central moving average (typically the SMA 20) and two bands at +/- 2 standard deviations. The bands expand with increasing volatility and contract with low volatility. The price tends to remain within the bands 95% of the time. A squeeze (contraction) heralds a breakout, while touching the outer bands may indicate overbought/oversold or trend continuation.

Syntax

var BBands(vars Series);
// or with specific periodvar BBands(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the BBands indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate BBandsvars Price = series(priceClose());var value = BBands(Price);var prevvalue = BBands(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "BBands", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.1.6. BOP - Balance Of Power

Description

An indicator that measures the balance between buyers and sellers, calculated as (Close - Open) / (High - Low). Positive values indicate dominant buying pressure, negative values indicate selling pressure. Fluctuating between -1 and +1, persistent extreme values suggest a strong supply/demand imbalance. Useful for confirming trend strength and identifying accumulation/distribution.

Syntax

```
var BOP(vars Series);
// or with specific periodvar BOP(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the BOP indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour
```

```
// Calculate BOPvars Price = series(priceClose());var value = BOP(Price);var prevvalue = BOP(Price)[1];
// Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "BOP", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.1.7. CCI - Commodity Channel Index

Description

An oscillator that measures the price's deviation from its statistical mean. Calculated as (Typical Price - SMA) / (0.015 * Mean Deviation). Typically fluctuating between -100 and +100, values above indicate extreme conditions. Above +100 suggests overbought, below -100 oversold. Effective for identifying cyclical reversals and unsustainable trends, originally developed for commodities but applicable to all markets.

Syntax

```
var CCI(vars Series);
// or with a specific period var CCI(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the CCI indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate CCIvars Price = series(priceClose());var value = CCI(Price);var prevvalue = CCI(Price)[1]; //
Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "CCI", value);
```

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible long signalif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible short signalif
(!NumOpenShort)
enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.1.8. CI - Choppiness Index

Description

Measures how choppy (sideways/random) the market is versus trending, with values from 0 to 100. Above 61.8 indicates a choppy/range-bound market, below 38.2 indicates a strong trend. Calculated using ATR and total range over one period. Essential for adaptive strategies: use mean reversion when CI is high, trend following when it is low. Prevents false signals from trend indicators in sideways markets.

Syntax

var CI(vars Series);
// or with a specific period var CI(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the CI indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate CIvars Price = series(priceClose());var value = CI(Price);var prevvalue = CI(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "CI", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {
// Positive crossover - possible long signalif
(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible short signalif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.1.9. ConnorsRSI - Connors RSI Indicator

Description

A composite system that combines three components: classic RSI, streak RSI (consecutive ups and downs), and ROC percentile rank. It produces more extreme and mean-reverting 0-100 values than the standard RSI. Above 90, it indicates strong overbought (short opportunities); below 10, it indicates strong oversold (long opportunities). Particularly effective for short-term mean-reversion trading on equities and ETFs.

Syntax

var ConnorsRSI(vars Series);
// or with a specific period var ConnorsRSI(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ConnorsRSI indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // 1-hour timeframe

// Calculate ConnorsRSIvars Price = series(priceClose());var value = ConnorsRSI(Price);var prevvalue = ConnorsRSI(Price)[1]; // Previous value

// Log of value
if(is(LOGFILE))printf("\n%s: %.4f", "ConnorsRSI", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Useful for identifying overbought/oversold conditions

1.1.10. CTI - Correlation Trend Indicator

Description

Measures trend strength using price-time correlation. High positive correlation indicates a strong uptrend, high negative correlation indicates a strong downtrend, and low correlation indicates a choppy/sideways market. Superior to traditional directional indicators in distinguishing true trends from consolidations. Useful for filtering: trade only when the CTI exceeds a significant correlation threshold.

Syntax

var CTI(vars Series);
// or with specific periodvar CTI(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the CTI indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate CTIvars Price = series(priceClose());var value = CTI(Price);var prevvalue = CTI(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "CTI", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.1.11. DX - Directional Movement Index

Description

A component of the ADX system that measures the normalized difference between +DI and -DI. High values indicate strong directionality (trend), low values indicate indecision or consolidation. It does not indicate the direction of the trend, but only the strength of the directional trend. Used as a filter: trade only when DX crosses the threshold, indicating the presence of an exploitable trend.

Syntax

var DX(vars Series);
// or with specific periodvar DX(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the DX indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate DXvars Price = series(priceClose());var value = DX(Price);var prevvalue = DX(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "DX", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.1.12. Ichimoku - Ichimoku Indicator

Description

Complete Japanese trading system with five components: Tenkan (conversion line), Kijun (base line), Senkou A and B (leading spans that form clouds), and Chikou (lagging line). Prices above the cloud are bullish, below are bearish. Thickness of the cloud indicates strong support/resistance. Tenkan cross Kijun generates signals. Complete system provides entry, stop, and target levels in a single framework.

Syntax

var Ichimoku(vars Series);
// or with specific periodvar Ichimoku(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Ichimoku indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Ichimokuvars Price = series(priceClose());var value = Ichimoku(Price);var prevvalue = Ichimoku(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Ichimoku", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced parameters.

Ichimoku

- Always test on historical data before using in production

1.1.13. Keltner - Keltner Channel

Description

Band system consisting of a central EMA ± N times the ATR. Similar to Bollinger but uses ATR instead of standard deviation. Bands expand/contract with volatility. Breakouts above the upper band are bullish, while breaks below the lower band are bearish. Prices tend to mean-revert toward the center in the absence of a trend. Squeeze (narrow bands) heralds an imminent breakout.

Syntax

var Keltner(vars Series);
// or with specific period var Keltner(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Keltner indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Keltnervars Price = series(priceClose());var value = Keltner(Price);var prevvalue = Keltner(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Keltner", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible long signalif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible short signalif
(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.1.14. MACD - Moving Average Convergence/Divergence

Description

Momentum oscillator formed by the difference between the fast EMA (12) and the slow EMA (26), with a signal line (EMA 9 of the MACD). MACD/signal crossovers generate buy/sell signals. MACD above/below zero indicates momentum direction. Divergences between MACD and price are powerful signals of impending reversion. One of the most popular indicators, versatile for trend-following and reversions.

Syntax

var MACD(vars Series);
// or with specific periodvar MACD(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MACD indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // 1-hour timeframe

// Calculate MACD vars Price = series(priceClose());var value = MACD(Price);var prevvalue = MACD(Price)[1]; // Previous value

// Log of value if(is(LOGFILE)) printf("\n%s: %.4f", "MACD", value);

// Example trading condition
if(value > 0 && prevvalue <= 0) {
// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) { // Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends
- Divergences with price can signal impending reversals

1.1.15. MACDExt - MACD with Various MA Types

Description

An extended version of the MACD that allows you to specify the type of moving average (SMA, EMA, WMA, DEMA, etc.) for fast, slow, and signal lines instead of just the EMA. Greater flexibility for customization. Some traders prefer TEMA or DEMA to further reduce lag. It allows testing of MACD variations for optimization on specific assets.

Syntax

```
var MACDExt(vars Series);
// or with specific periodvar MACDExt(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MACDExt indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // 1-hour timeframe

// Calculate MACDExtvars Price = series(priceClose());
var value = MACDExt(Price);
var prevvalue = MACDExt(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "MACDExt", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced MACDExt parameters.
- Always test on historical data before using in production

1.1.16. MACDFix - MACD with Standard Parameters

Description

MACD implementation with standard fixed parameters (12, 26, 9) without customization options. "No-frills" version to simplify implementation when using classic parameters. Computationally more efficient than the customizable version. Useful when you want standard MACD without parameter-passing overhead.

Syntax

```
var MACDFix(vars Series);
// or with specific periodvar MACDFix(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MACDFix indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate MACDFixvars Price = series(priceClose());var value = MACDFix(Price);var prevvalue = MACDFix(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "MACDFix", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced MACDFix parameters.
- Always test on historical data before using in production

1.1.17. MMI - Market Meanness Index

Description

Quantifies how much the market is mean-reverting on a 0-100 scale. High values (>75) indicate strong mean-reversion favorable for contrarian strategies, low values (<25) indicate momentum/trending favorable for trend-following. Calculated by comparing recent movements with historical distribution. Advanced filter for strategy selection based on market regime.

Syntax

var MMI(vars Series);
// or with a specific period var MMI(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MMI indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

```
// Calculate MMIvars Price = series(priceClose());var value = MMI(Price);var prevvalue = MMI(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "MMI", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.1.18. MinusDI - Minus Directional Indicator

Description

Component of the ADX system that measures bearish pressure. High values indicate strong downward pressure. Compared to PlusDI: when MinusDI > PlusDI suggests a downtrend, crossover generates sell signals. Used in conjunction with ADX to confirm trends: High ADX + MinusDI > PlusDI = confirmed strong downtrend.

Syntax

```
var MinusDI(vars Series);
// or with specific periodvar MinusDI(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MinusDI indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate MinusDIvars Price = series(priceClose());var value = MinusDI(Price);var prevvalue = MinusDI(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "MinusDI", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
```

longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.1.19. MinusDM - Minus Directional Movement

Description

Measures how much of a bar's movement is downward. Part of the MinusDI calculation. If current low < previous low, MinusDM = previous low - current low, otherwise 0. Quantifies raw downward pressure before smoothing and normalization in MinusDI.

Syntax

var MinusDM(vars Series);
// or with specific periodvar MinusDM(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MinusDM indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate MinusDMvars Price = series(priceClose());var value = MinusDM(Price);var prevvalue = MinusDM(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "MinusDM", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced MinusDM parameters

- Always test on historical data before using in production

1.1.20. OBV - On Balance Volume

Description

Cumulative indicator that adds volume when the price closes up and subtracts it when it closes down. Theory: Volume precedes price. Rising OBV confirms an uptrend, falling confirms a downtrend. Divergences between OBV and price are powerful early warnings: price new high but OBV not confirming suggests a weak trend headed for a reversal.

Syntax

var OBV(vars Series);
// or with specific periodvar OBV(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the OBV indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // 1-hour timeframe

// Calculate OBVvars Price = series(priceClose());var value = OBV(Price);var prevvalue = OBV(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "OBV", value);

// Example trading condition
if(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends
- Divergences with price can signal impending reversals
- Incorporates volume information for more comprehensive analysis

1.1.21. PlusDI - Plus Directional Indicator

Description

ADX component that measures bullish pressure. High values indicate strong upward pressure. When PlusDI > MinusDI, it suggests an uptrend. A crossover of PlusDI above MinusDI generates a buy signal. Used with ADX: High ADX + PlusDI > MinusDI = confirmed strong uptrend, optimal conditions for long entries.

Syntax

var PlusDI(vars Series);
// or with specific periodvar PlusDI(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the PlusDI indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate PlusDIvars Price = series(priceClose());var value = PlusDI(Price);var prevvalue = PlusDI(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "PlusDI", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.1.22. PlusDM - Plus Directional Movement

Description

Measures how much of a bar's movement is upward. Part of the PlusDI calculation. If current high > previous high, PlusDM = current high - previous high, otherwise 0. Quantifies raw upward pressure before smoothing and normalization in PlusDI. Foundation for directional movement systems.

Syntax

var PlusDM(vars Series);
// or with specific periodvar PlusDM(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the PlusDM indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate PlusDMvars Price = series(priceClose());var value = PlusDM(Price);var prevvalue = PlusDM(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "PlusDM", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced PlusDM parameters
- Always test on historical data before using in production

1.1.23. RSI - Relative Strength Index (Original)

Description

Momentum oscillator 0-100 that compares the magnitude of recent gains vs. losses. RSI = 100 - (100 / (1 + RS)), where RS = average gain / average loss over N periods (14 defaults). Above 70, overbought; below 30, oversold. Divergences are powerful signals. One of the most popular indicators, versatile for reversals and momentum analysis.

Syntax

var RSI(vars Series);
// or with a specific period var RSI(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the RSI indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // 1-hour timeframe

// Calculate RSIvars Price = series(priceClose());var value = RSI(Price);var prevvalue = RSI(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "RSI", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Useful for identifying overbought/oversold conditions
- Divergences with price can signal impending reversals

1.1.24. RSIS - RSI Simple MA

Description

A variant of the RSI that uses a simple moving average instead of an exponential one to calculate average gains/losses. Less reactive to recent changes, smoother. Generates fewer false signals but also less timely. Preferred by some traders to reduce whipsaw in choppy markets. Tradeoff between responsiveness and stability.

Syntax

var RSIS(vars Series);
// or with specific periodvar RSIS(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the RSIS indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate RSISvars Price = series(priceClose());var value = RSIS(Price);var prevvalue = RSIS(Price)[1];
// Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "RSIS", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)
enterShort();
}}
```

Usage Notes

- See the Zorro documentation for advanced RSIS parameters
- Always test on historical data before using in production

1.1.25. SAR - Parabolic SAR

Description

Stop-and-reverse system that provides trailing stops that accelerate with the trend. Dots appear above the price in a downtrend (short), below in an uptrend (long). When the price touches the SAR level, the position reverses. Acceleration factor progressively increases, making the stop more aggressive. Designed to always be in the market, capturing trending moves.

Syntax

```
var SAR(vars Series);
// or with specific periodvar SAR(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the SAR indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate SARvars Price = series(priceClose());var value = SAR(Price);var prevvalue = SAR(Price)[1];
// Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "SAR", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {
// Positive crossover - possible long signalif
(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible short
signalif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.1.26. Stoch - Stochastic Oscillator

Description

Compare close position within recent high-low range: (%K = (Close - LL) / (HH - LL)) * 100. %D is moving average of %K. Values 0-100. Above 80 overbought, below 20 oversold. %K/%D crossovers generate signals. Divergences very powerful. One of most popular mean-reversion indicators, especially effective in ranging markets.

Syntax

```
var Stoch(vars Series);
// or with specific periodvar Stoch(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Stoch indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour
```

```
// Calculate Stochvars Price = series(priceClose());var value = Stoch(Price);var prevvalue = Stoch(Price)[1]; // Previous value

// Log of value
if(is(LOGFILE))printf("\n%s: %.4f", "Stoch", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Useful for identifying overbought/oversold conditions
- Divergences with price can signal impending reversals

1.1.27. StochF - Stochastic Fast

Description

Raw stochastic (%K) without %D smoothing. More reactive but noisier. Generates more signals but also more false signals. Preferred by short-term traders who need responsiveness. Trade-off: speed vs reliability. Useful in fast-moving markets where every tick matters, less in slower markets where noise dominates.

Syntax

```
var StochF(vars Series);
// or with specific periodvar StochF(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the StochF indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate StochFvars Price = series(priceClose());var value = StochF(Price);var prevvalue = StochF(Price)[1]; // Previous value
```

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "StochF", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced StochF parameters
- Always test on historical data before using in production

1.1.28. StochRSI - Stochastic RSI

Description

Applying the stochastic formula to the RSI instead of the price. Double normalization makes the 0-100 oscillator much more overbought/oversold than the RSI alone. Above 80, extremely overbought; below 20, extremely oversold. More extreme readings than the plain RSI, making it more suitable for mean reversion. Combination of two powerful indicators.

Syntax

var StochRSI(vars Series);
// or with a specific period var StochRSI(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the StochRSI indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // 1-hour timeframe

// Calculate StochRSIvars Price = series(priceClose());var value = StochRSI(Price);var prevvalue = StochRSI(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "StochRSI", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Useful for identifying overbought/oversold conditions

1.1.29. TSI - Trend Strength Index

Description

Measures trend strength by combining direction and magnitude of movement. High absolute values indicate strong trends, while values close to zero indicate weak/no trend. It can distinguish strong trends (high TSI) from weak drifts (low TSI). Useful filter for trading only when trends are strong enough to overcome the friction of entering/exiting.

Syntax

var TSI(vars Series);
// or with a specific period var TSI(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the TSI indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate TSIvars Price = series(priceClose());var value = TSI(Price);var prevvalue = TSI(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "TSI", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.1.30. UltOsc - Ultimate Oscillator

Description

Multi-timeframe oscillator combining 7-, 14-, and 28-period momentum to reduce false signals. Weighted combination of short, medium, and long-term buying pressure. Values 0-100. Above 70, overbought; below 30, oversold. Very reliable divergences. More robust than single-timeframe oscillators because cross-timeframe confirmation is required.

Syntax

var UltOsc(vars Series);
// or with specific periodvar UltOsc(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the UltOsc indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate UltOscvars Price = series(priceClose());var value = UltOsc(Price);var prevvalue = UltOsc(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "UltOsc", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort) enterShort();
}}

Usage Notes

- Useful for identifying overbought/oversold conditions
- Divergences with price can signal impending reversals

1.2. 📈 MOVING AVERAGES

1.2.1. ALMA - Arnaud Legoux Moving Average

Description

Gaussian moving average that reduces lag while maintaining smoothness. It uses a configurable offset and sigma to balance responsiveness and noise reduction. Superior to traditional moving averages for following trends with less delay, particularly effective at reducing false signals during consolidations while maintaining responsiveness to significant movements.

Syntax

var ALMA(vars Series);
// or with a specific period var ALMA(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ALMA indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ALMAvars Price = series(priceClose());var value = ALMA(Price);var prevvalue = ALMA(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ALMA", value);

// Example trading condition
if(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.2.2. AvgPrice - Average Price

Description

Calculates the average price of a candlestick as (Open + High + Low + Close) / 4. It provides a more complete representation of price action than the close alone, incorporating all the information from the bar. Useful as a basis for smoother indicator calculations and for identifying the true center of gravity of daily price action.

Syntax

var AvgPrice(vars Series);
// or with a specific period var AvgPrice(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the AvgPrice indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate AvgPrice
vars Price = series(priceClose());
var value = AvgPrice(Price);var prevvalue = AvgPrice(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "AvgPrice", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced AvgPrice settings.
- Always test on historical data before using in production

1.2.3. DEMA - Double Exponential Moving Average

Description

A doubly smoothed moving average that offers less lag than the standard EMA. Calculated as 2*EMA - EMA(EMA), it reacts more quickly to price changes. Superior to SMA and EMA in its balance between smoothness and responsiveness. Fast/slow DEMA crossovers generate more timely signals, useful for trend-following that requires rapid entries.

Syntax

var DEMA(vars Series);
// or with specific periodvar DEMA(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the DEMA indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate DEMAvars Price = series(priceClose());var value = DEMA(Price);var prevvalue = DEMA(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "DEMA", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.2.4. EMA - Exponential Moving Average

Description

Moving average that gives more weight to recent prices with exponential decay. More reactive than the SMA, it reduces lag while maintaining smoothness. Key parameter: lookback period (e.g., 12, 26, 200 days). A fundamental building block for countless indicators (MACD, PPO) and strategies. Crossovers of multiple EMAs (golden/death crosses) are classic trend change signals.

Syntax

var EMA(vars Series);
// or with specific periodvar EMA(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the EMA indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // 1-hour timeframe

// Calculate EMAvars Price = series(priceClose());var value = EMA(Price);var prevvalue = EMA(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "EMA", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.2.5. HMA - Hull Moving Average

Description

Highly responsive moving average that nearly eliminates lag using weighted moving averages with a square root period. Calculated as WMA (2*WMA(n/2) - WMA(n)) with a period of sqrt(n). Excellent balance between smoothing and responsiveness, superior to EMA. Crossovers generate timely signals, while the curve maintains smoothness. Ideal for trend-following that requires speed without compromising signal quality.

Syntax

var HMA(vars Series);
// or with a specific period var HMA(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the HMA indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate HMAvars Price = series(priceClose());var value = HMA(Price);var prevvalue =
HMA(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "HMA", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();
}
}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.2.6. HTTrendline - Hilbert Transform Instantaneous Trendline

Description

Calculate an instant trendline using the Hilbert Transform, following the trend with minimal lag. More responsive than traditional moving averages, it quickly adapts to trend changes. Price crossovers with the trendline generate signals. Superior filtering of the trend component from the cyclical one, maintaining smoothness despite adaptivity.

Syntax

```
var HTTrendline(vars Series);
// or with specific periodvar HTTrendline(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the HTTrendline indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour
```

```
// Calculate HTTrendlinevars Price = series(priceClose());var value = HTTrendline(Price);var prevvalue = HTTrendline(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "HTTrendline", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)
enterLong();
}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.2.7. KAMA - Kaufman Adaptive Moving Average

Description

A moving average that varies its smoothing constant based on the efficiency ratio. In trending markets, it becomes faster (less lag), and in choppy markets, slower (more smoothing). Self-adjusting for market conditions, it reduces whipsaw while maintaining responsiveness when needed. Superior to a fixed-period MA in balancing the contradictory requirements of fast response and noise reduction.

Syntax

```
var KAMA(vars Series);
// or with specific periodvar KAMA(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the KAMA indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate KAMAvars Price = series(priceClose());var value = KAMA(Price);var prevvalue = KAMA(Price)[1]; // Previous value
```

```
// Log of value
if(is(LOGFILE))
printf("\n%s: %.4f", "KAMA", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.2.8. KAMA2 - KAMA with Individual Settings

Description

An extended version of KAMA that allows separate customization of the fast and slow periods instead of using defaults. Greater flexibility for tuning to specific markets or timeframes. It retains KAMA's adaptive logic but with more control over the maximum and minimum adaptation speeds. Useful for fine-tuning in backtesting.

Syntax

```
var KAMA2(vars Series);
// or with specific periodvar KAMA2(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the KAMA2 indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate KAMA2
vars Price = series(priceClose());var value = KAMA2(Price);var prevvalue = KAMA2(Price)[1]; //
Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "KAMA2", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
```

longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced KAMA2 parameters
- Always test on historical data before using in production

1.2.9. Laguerre - Laguerre Lowpass Filter

Description

A sophisticated digital filter that offers superior smoothing with minimal lag using Laguerre polynomials. The gamma parameter (0-1) controls smoothness: low = more responsive, high = smoother. Excellent for trend identification, eliminating noise without excessive delay. Superior to simple moving averages, used in high-frequency strategies where lag is critical.

Syntax

var Laguerre(vars Series);
// or with a specific period var Laguerre(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Laguerre indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Laguerrevars Price = series(priceClose());var value = Laguerre(Price);var prevvalue = Laguerre(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Laguerre", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.2.10. LinearReg - Linear Regression

Description

Calculate the line of best fit through prices using least squares regression. Returns the value fitted to the last bar. The smoothing effect is similar to moving average but fits the linear trend better. Deviations from the regression line identify overbought/oversold conditions. The angle and slope of the regression line quantify trend strength and direction.

Syntax

var LinearReg(vars Series);
// or with specific periodvar LinearReg(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the LinearReg indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate LinearRegvars Price = series(priceClose());var value = LinearReg(Price);var prevvalue = LinearReg(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "LinearReg", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends
- Useful for identifying overbought/oversold conditions

1.2.11. LSMA - Least Squares Moving Average

Description

Moving average calculated via linear regression on prices. Each point is the value fitted to the regression line at the last bar of the period. Lower lag than the SMA because the regression line "anticipates" the trend direction. More responsive to trend changes while maintaining smoothness. Crossovers with price or another LSMA generate more timely signals than SMA crossovers.

Syntax

var LSMA(vars Series);
// or with specific periodvar LSMA(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the LSMA indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate LSMAvars Price = series(priceClose());var value = LSMA(Price);var prevvalue = LSMA(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "LSMA", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort) enterShort();
}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.2.12. MAMA - MESA Adaptive Moving Average

Description

Adaptive moving average that uses the MESA (Maximum Entropy Spectrum Analysis) algorithm to adapt speed to the dominant cycle. Two outputs: MAMA line and FAMA (following) line. Crossovers

generate signals. Automatically adjusts between fast and slow based on period detection. Superior at following trends of varying speeds without manual parameter adjustment.

Syntax

var MAMA(vars Series);
// or with a specific period var MAMA(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MAMA indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate MAMAvars Price = series(priceClose());var value = MAMA(Price);var prevvalue = MAMA(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "MAMA", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible long signalif
(!NumOpenLong)
enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible short signalif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.2.13. MovingAverage - Generic Moving Average

Description

Generic function that calculates moving averages of the specified type: SMA, EMA, WMA, DEMA, TEMA, KAMA, etc. Allows you to switch MA types without rewriting code. Useful for testing which MA type works best for a specific asset. Centralizes MA calculation, simplifying maintenance and allowing dynamic MA type selection in adaptive systems.

Syntax

var MovingAverage(vars Series);
// or with specific periodvar MovingAverage(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MovingAverage indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate MovingAveragevars Price = series(priceClose());var value = MovingAverage(Price);var prevvalue = MovingAverage(Price)[1]; // Previous value

// Log of value
if(is(LOGFILE))printf("\n%s: %.4f", "MovingAverage", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced MovingAverage parameters.
- Always test on historical data before using in production

1.2.14. MovingAverageVariablePeriod - MA with Variable Period

Description

Moving average where the period changes dynamically based on the input series instead of being fixed. Allows implementations where the period is a function of volatility, volume, or other factors. Example: use shorter periods in volatile markets, longer periods in calm markets. Advanced adaptive technique that goes beyond simple parameter optimization.

Syntax

var MovingAverageVariablePeriod(vars Series);
// or with specific periodvar MovingAverageVariablePeriod(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MovingAverageVariablePeriod indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate MovingAverageVariablePeriodvars Price = series(priceClose());var value = MovingAverageVariablePeriod(Price);var prevvalue = MovingAverageVariablePeriod(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "MovingAverageVariablePeriod", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Incorporates volume information for more comprehensive analysis

1.2.15. MedPrice - Center Price of Candle

Description

Calculates (High + Low) / 2 for each bar. Represents the midpoint of the bar's range, considered by many to be more representative of typical price action than the close alone. Less susceptible to manipulation of the close. Used as an input for various indicators instead of the close when you want to focus on the center of the range.

Syntax

```
var MedPrice(vars Series);
// or with a specific period var MedPrice(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MedPrice indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate MedPricevars Price = series(priceClose());var value = MedPrice(Price);var prevvalue = MedPrice(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "MedPrice", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced MedPrice parameters
- Always test on historical data before using in production

1.2.16. MidPoint - Center Value of Period

Description

Calculates (Highest + Lowest) / 2 of the last N periods. Identifies the center of the recent range. Often used as a baseline for mean reversion: when the price strays too far from the midpoint, it tends to revert. Provides a stable reference for assessing whether the price is extended to the top or bottom of the range.

Syntax

```
var MidPoint(vars Series);
// or with specific periodvar MidPoint(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MidPoint indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate MidPointvars Price = series(priceClose());var value = MidPoint(Price);var prevvalue = MidPoint(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "MidPoint", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced MidPoint parameters
- Always test on historical data before using in production

1.2.17. MidPrice - Center Price of Period

Description

Calculates (Highest High + Lowest Low) / 2 of the last N price periods. Similar to MidPoint, but specifically for prices rather than general values. Identifies the fair value or equilibrium level of the recent trading range. Price above MidPrice is in bull territory, below in bear territory.

Syntax

```
var MidPrice(vars Series);
// or with a specific period var MidPrice(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MidPrice indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate MidPricevars Price = series(priceClose());var value = MidPrice(Price);var prevvalue =
```

MidPrice(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "MidPrice", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();
}
}

Usage Notes

- See the Zorro documentation for advanced MidPrice parameters
- Always test on historical data before using in production

1.2.18. SMA - Simple Moving Average

Description

Simple arithmetic mean of prices over N periods. Each price has equal weight. Most basic smoothing tool. Significant lag but very smooth. Foundation for countless indicators. Crossovers of multiple SMAs (golden/death crosses) are classical signals. Despite simplicity, it remains widely used as a benchmark and for long-term trend identification.

Syntax

var SMA(vars Series);
// or with a specific period var SMA(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the SMA indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // 1-hour timeframe

// Calculate SMAvars Price = series(priceClose());var value = SMA(Price);var prevvalue = SMA(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "SMA", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible long signalif
(!NumOpenLong)
enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible short
signalif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.2.19. T3 - Triple Smoothed MA

Description

Moving average with triple exponential smoothing for superior smoothness and minimal lag. Uses volume factor coefficient for control lag vs smoothness trade-off. Smoother of THEME while maintaining responsiveness. Excellent for trend identification eliminating false swings. Popular for crossover systems requiring very clean signals.

Syntax

var T3(vars Series);
// or with specific periodvar T3(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the T3 indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate T3vars Price = series(priceClose());var value = T3(Price);var prevvalue = T3(Price)[1]; // Previous value

// Log of value
if(is(LOGFILE))printf("\n%s: %.4f", "T3", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends
- Incorporates volume information for more comprehensive analysis

1.2.20. THEME - Triple EMA

Description

Triple smoothed EMA calculated as 3EMA - 3EMA(EMA) + EMA(EMA(EMA)). Reduces lag even further than DEMA while maintaining smoothness. Very responsive to trend changes. Trade-off: more responsiveness also means potentially more false signals in choppy markets. Used when speed is at a premium.

Syntax

var THEME(vars Series);
// or with specific periodvar TEMA(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the TEMA indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate TEMAvars Price = series(priceClose());var value = TEMA(Price);var prevvalue = TEMA(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "TEMA", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.2.21. Trima - Triangular Moving Average

Description

A moving average that gives triangular weighting: more weight to the center of the period, less to the extremes. Produced by taking an SMA of an SMA. Double smoothing results in a very smooth curve with less weighted endpoints. Less common than an SMA/EMA but useful when you prefer to deemphasize the oldest/newest data in favor of the middle range.

Syntax

var Trima(vars Series);
// or with a specific period var Trima(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Trima indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Trimavars Price = series(priceClose());var value = Trima(Price);var prevvalue = Trima(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Trima", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced Trima parameters
- Always test on historical data before using in production

1.2.22. TSF - Time Series Forecast

Description

Projection based on linear regression: extends the regression line a forward period as a next value forecast. Simple predictive tool assuming trend continuation. Output is expected price next bar based on recent linear trend. More timely than moving averages being forward-looking rather than backward-looking.

Syntax

var TSF(vars Series);
// or with specific periodvar TSF(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the TSF indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate TSFvars Price = series(priceClose());var value = TSF(Price);var prevvalue = TSF(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "TSF", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.2.23. TypPrice - Typical Price

Description

Calculates (High + Low + Close) / 3. Representative point that incorporates more information about the candlestick than just the close or midpoint. Widely used as the basis for pivot calculations and for indicators that capture more complete price action of the bar. More robust to abnormal closes.

Syntax

var TypPrice(vars Series);
// or with a specific period var TypPrice(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the TypPrice indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate TypPricevars Price = series(priceClose());var value = TypPrice(Price);var prevvalue = TypPrice(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "TypPrice", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();
}
}

Usage Notes

- See the Zorro documentation for advanced TypPrice parameters.
- Always test on historical data before using in production

1.2.24. WCLPrice - Weighted Close Price

Description

Calculate (High + Low + Close*2) / 4. Gives double weight to the close compared to the high/low, reflecting the importance of where the day ended. Representative price that emphasizes the closing of the auction. Used in some technical indicators instead of the simple close or typical price when the close is considered more significant.

Syntax

var WCLPrice(vars Series);
// or with specific periodvar WCLPrice(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the WCLPrice indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate WCLPricevars Price = series(priceClose());var value = WCLPrice(Price);var prevvalue = WCLPrice(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "WCLPrice", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)
enterLong();
}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced WCLPrice parameters.
- Always test on historical data before using in production

1.2.25. WMA - Weighted Moving Average

Description

Moving average with linear weights: most recent price has highest weight, progressively decreasing for older prices. More responsive than SMA, less than EMA. Good balance between responsiveness and smoothing. Weight pattern is linear vs exponential of EMA. Used when linear weighting scheme preferred over exponential.

Syntax

var WMA(vars Series);
// or with specific periodvar WMA(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the WMA indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate WMAvars Price = series(priceClose());var value = WMA(Price);var prevvalue =
WMA(Price)[1]; // Previous value

// Log of value
if(is(LOGFILE))printf("\n%s: %.4f", "WMA", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced WMA parameters.
- Always test on historical data before using in production

1.2.26. ZMA - Zero-Lag Moving Average

Description

Moving average that virtually eliminates lag using error-correction feedback. Calculates expected value and corrects for difference between EMA and expected. Extremely responsive maintaining smoothness. Among fastest MAs without excessively compromising stability. Ideal for strategies requiring immediate response to trend changes.

Syntax

```
var ZMA(vars Series);
// or with specific periodvar ZMA(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ZMA indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ZMAvars Price = series(priceClose());var value = ZMA(Price);var prevvalue =
ZMA(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ZMA", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.3. 🔄 OSCILLATORS

1.3.1. AC - Accelerator Oscillator

Description

A technical indicator that measures the acceleration of price momentum. It is calculated as the difference between the 5-period Awesome Oscillator and a 34-period moving average. Positive values indicate bullish acceleration, negative values indicate bearish acceleration. Useful for confirming trends and identifying potential momentum reversals.

Syntax

```
var AC(vars Series);
// or with specific periodvar AC(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the AC indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ACvars Price = series(priceClose());var value = AC(Price);var prevvalue = AC(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "AC", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.3.2. ADO - Accumulation/Distribution Oscillator

Description

An oscillator that combines price and volume to determine whether an asset is accumulating (demand) or distributing (supply). It calculates the relationship between the close and the daily range multiplied by volume. Increasing values indicate accumulation (bullish), decreasing values indicate distribution (bearish). Particularly useful for confirming breakouts and identifying divergences.

Syntax

var ADO(vars Series);
// or with a specific period var ADO(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ADO indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

```
// Calculate ADOvars Price = series(priceClose());var value = ADO(Price);var prevvalue = ADO(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ADO", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Divergences with price can signal impending reversals
- Incorporates volume information for more comprehensive analysis

1.3.3. AO - Awesome Oscillator

Description

Momentum oscillator that measures the difference between a 5-period moving average and a 34-period moving average of the midpoints (High + Low)/2. Positive values above zero indicate bullish momentum, negative values indicate bearish momentum. Zero line crossovers generate trading signals, while patterns such as "twin peaks" and "saucers" indicate potential trend reversals.

Syntax

```
var AO(vars Series);
// or with a specific period var AO(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the AO indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate AOvars Price = series(priceClose());var value = AO(Price);var prevvalue = AO(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "AO", value);
```

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible long signalif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible short signalif
(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.3.4. APO - Absolute Price Oscillator

Description

Absolute difference in dollars between two exponential moving averages (typically 12 and 26 periods). Similar to the MACD but expressed as an absolute value rather than a percentage, making it more suitable for comparing assets with significantly different prices. Crossovers above/below zero indicate changes in momentum, with divergences signaling potential reversals.

Syntax

var APO(vars Series);
// or with specific periodvar APO(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the APO indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate APOvars Price = series(priceClose());var value = APO(Price);var prevvalue = APO(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "APO", value);

// Example trading condition
if(value > 0 && prevvalue <= 0) {
// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Divergences with price can signal impending reversals

1.3.5. AroonOsc - Aroon Oscillator

Description

Difference between Aroon Up and Aroon Down, ranging between -100 and +100. Positive values indicate an uptrend, negative values indicate a downtrend. Crossovers of the zero line generate trend reversal signals. Extreme values (+100/-100) indicate very strong trends, while values close to zero suggest a directionless market or a consolidation phase.

Syntax

var AroonOsc(vars Series);
// or with a specific period var AroonOsc(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the AroonOsc indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate AroonOscvars Price = series(priceClose());var value = AroonOsc(Price);
var prevvalue = AroonOsc(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "AroonOsc", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.3.6. BBOsc - Bollinger Bands Oscillator

Description

Transforms Bollinger Bands into a normalized oscillator between 0 and 100, measuring the price's position relative to the bands. Values above 80 indicate a price close to the upper band (potential overbought), while values below 20 indicate a price close to the lower band (oversold). Easier to interpret than traditional bands and useful for multi-timeframe comparisons.

Syntax

var BBOsc(vars Series);
// or with a specific period var BBOsc(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the BBOsc indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate BBOscvars Price = series(priceClose());var value = BBOsc(Price);var prevvalue = BBOsc(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "BBOsc", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced BBOsc parameters
- Always test on historical data before using in production

1.3.7. CGOsc - Center Of Gravity Oscillator

Description

Oscillator that identifies the center of gravity of recent prices using a weighted algorithm. It anticipates turning points by identifying where the price "mass" is concentrated. Values that reach

extremes and then reverse provide signals of potential trend reversal. Less lag than traditional oscillators, useful for precise entries in mean-reversion strategies.

Syntax

var CGOsc(vars Series);
// or with specific periodvar CGOsc(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the CGOsc indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate CGOscvars Price = series(priceClose());var value = CGOsc(Price);var prevvalue = CGOsc(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "CGOsc", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.3.8. CMO - Chande Momentum Oscillator

Description

Momentum oscillator bounded between -100 and +100 that measures trend strength using the sum of gains and losses. Above +50 indicates strong bullish momentum, below -50 strong bearish momentum. Less smoothed than the RSI, more reactive to changes. Extreme values suggest overbought/oversold but also possible trend strength. Divergences with price are powerful reversal signals.

Syntax

var CMO(vars Series);
// or with a specific period var CMO(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the CMO indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate CMOvars Price = series(priceClose());var value = CMO(Price);var prevvalue = CMO(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "CMO", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends
- Useful for identifying overbought/oversold conditions
- Divergences with price can signal impending reversals

1.3.9. DCOsc - Donchian Channel Oscillator

Description

Transforms the Donchian Channel into a normalized oscillator that shows the price's position within the channel, from 0 (at the low) to 100 (at the high). Above 80 indicates the price is at the top of the range (possible overbought), while below 20 indicates the price is at the bottom (oversold). More versatile than the raw bands, it allows for multi-timeframe comparisons and the identification of divergences with momentum indicators.

Syntax

var DCOsc(vars Series);
// or with specific periodvar DCOsc(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the DCOsc indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate DCOscvars Price = series(priceClose());var value = DCOsc(Price);var prevvalue = DCOsc(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "DCOsc", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Useful for identifying overbought/oversold conditions
- Divergences with price can signal impending reversals

1.3.10. DPO - Detrended Price Oscillator

Description

It removes the trend from prices by centering the moving average in the middle of the lookback period instead of the last point. It isolates cyclical oscillations from the underlying trend. Positive/negative values do not indicate trend direction but rather the cyclical phase. Useful for identifying overbought/oversold conditions within the trend and for cycle analysis that requires separating the trend from the cyclical component.

Syntax

var DPO(vars Series);
// or with a specific period var DPO(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the DPO indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate DPOvars Price = series(priceClose());var value = DPO(Price);var prevvalue =
DPO(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "DPO", value);

// Example trading condition
if(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends
- Useful for identifying overbought/oversold conditions

1.3.11. ER - Efficiency Ratio

Description

Measures the efficiency of price movement: how much net movement vs. total movement. Values close to 1 indicate a strong directional trend, while values close to 0 indicate choppy sideways action. Calculated as (Net Change / Sum of Absolute Changes). Used for adaptive indicators that accelerate in trending markets and slow down in choppy markets (e.g., KAMA). Identifies when breakouts are genuine vs. false.

Syntax

```
var ER(vars Series);
// or with specific periodvar ER(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ER indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ER
vars Price = series(priceClose());var value = ER(Price);var prevvalue = ER(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ER", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.3.12. Mom - Momentum

Description

Simply calculate the difference between the current price and the price N periods ago: Close - Close[N]. Positive indicates upward momentum, negative indicates downward momentum. Raw momentum without smoothing, very responsive but noisy. Foundation for many more sophisticated oscillators. Crossing zero indicates a shift in the direction of momentum.

Syntax

```
var Mom(vars Series);
// or with specific periodvar Mom(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Mom indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Momvars Price = series(priceClose());var value = Mom(Price);var prevvalue = Mom(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Mom", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced Mom parameters.
- Always test on historical data before using in production

1.3.13. PPO - Percentage Price Oscillator

Description

Similar to the MACD but expressed as a percentage of the price instead of absolute values. Calculated as ((EMA12 - EMA26) / EMA26) * 100. It allows you to compare momentum across different prices and assets. Positive values indicate a fast MA above a slow (bullish) one, negative values indicate the opposite. Divergences and zero-line crossings generate trading signals like the MACD.

Syntax

var PPO(vars Series);
// or with a specific period var PPO(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the PPO indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate PPOvars Price = series(priceClose());var value = PPO(Price);var prevvalue = PPO(Price)[1];

```
// Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "PPO", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Divergences with price can signal impending reversals

1.3.14. ROC - Rate of Change

Description

Calculates the percentage change between the current price and N bars ago. Positive values indicate upward momentum, negative values indicate downward momentum. Extreme values indicate overbought/oversold. Zero line crossings signal momentum shifts. Divergences between ROC and price predict reversals. A simple yet effective fundamental momentum indicator.

Syntax

```
var ROC(vars Series);
// or with a specific period var ROC(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ROC indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ROCvars Price = series(priceClose());var value = ROC(Price);var prevvalue =
ROC(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ROC", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}
```

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();
}
}

Usage Notes

- Useful for identifying overbought/oversold conditions
- Divergences with price can signal impending reversals

1.3.15. ROCP - Rate of Change Percentage

Description

Identical to ROC but explicitly expressed as a percentage: ((price - price[n]) / price[n]) * 100. The percentage format makes it more intuitive for interpretation and comparison across assets. Same usefulness as ROC for momentum analysis, overbought/oversold identification, and divergence detection.

Syntax

var ROCP(vars Series);
// or with specific periodvar ROCP(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of ROCP indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ROCPvars Price = series(priceClose());var value = ROCP(Price);var prevvalue = ROCP(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ROCP", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)
enterLong();
}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Useful for identifying overbought/oversold conditions
- Divergences with price can signal impending reversals

1.3.16. ROCR - Rate of Change Ratio

Description

Rate of change expressed as a ratio instead of a percentage: price / price[n]. Values above 1 indicate a gain, below 1 a loss. Example: 1.05 = 5% gain, 0.95 = 5% loss. Useful in calculations where the ratio format is more convenient than the percentage. Identical information to ROC but different representation.

Syntax

var ROCR(vars Series);
// or with specific periodvar ROCR(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ROCR indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ROCRvars Price = series(priceClose());var value = ROCR(Price);var prevvalue = ROCR(Price)[1]; // Previous value

// Log of value
if(is(LOGFILE))
printf("\n%s: %.4f", "ROCR", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced ROCR parameters
- Always test on historical data before using in production

1.3.17. ROCL - Logarithmic Return

Description

Natural logarithm of ROCR: ln(price / price[n]). Log returns are additive (convenient for calculating cumulative returns) and more normally distributed than simple returns. Preferred in statistical analysis and modeling. Small log returns approximate percentage returns but have better statistical properties for time-series analysis.

Syntax

var ROCL(vars Series);
// or with a specific period var ROCL(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of ROCL indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ROCLvars Price = series(priceClose());var value = ROCL(Price);var prevvalue = ROCL(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ROCL", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced ROCL parameters
- Always test on historical data before using in production

1.3.18. ROCR100 - Rate of Change Ratio 100 Scale

Description

ROCR multiplied by 100 for convenience. Values above 100 indicate gain, below 100 indicate loss. Example: 105 = 5% gain. A 100-based scale makes tracking more intuitive. Identical information to ROCR but different scaling. Some traders find this representation easier to visualize on charts.

Syntax

var ROCR100(vars Series);
// or with specific periodvar ROCR100(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ROCR100 indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ROCR100vars Price = series(priceClose());var value = ROCR100(Price);var prevvalue = ROCR100(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ROCR100", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced ROCR100 parameters
- Always test on historical data before using in production

1.3.19. RVI - Relative Vigor Index

Description

Compare closing range with total range: close vs. open momentum (vigor) relative to the high-low range. Positive values indicate close tends above open (bullish vigor), negative opposites. Smoothed

with SMA. Crossovers of RVI with signal lines generate trading signals. Measures market movement conviction.

Syntax

var RVI(vars Series);
// or with a specific period var RVI(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the RVI indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate RVIvars Price = series(priceClose());var value = RVI(Price);var prevvalue = RVI(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "RVI", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced RVI parameters
- Always test on historical data before using in production

1.3.20. SIROC - Smoothed Rate of Change

Description

ROC after smoothing with moving average to reduce noise. Less whipsaw than the raw ROC but maintaining momentum information. Various smoothing lengths possible. Zero line crossings generate signals. Divergences are less frequent but more reliable. Preferred when the standard ROC produces too many false signals.

Syntax

var SIROC(vars Series);
// or with a specific period var SIROC(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the SIROC indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate SIROCvars Price = series(priceClose());var value = SIROC(Price);var prevvalue = SIROC(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "SIROC", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible long signalif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible short signalif
(!NumOpenShort)enterShort();}}

Usage Notes

- Divergences with price can signal impending reversals

1.3.21. SMom - Smoothed Momentum

Description

Momentum indicator after smoothing to reduce noise. More stable than raw momentum, retaining information on the velocity of price change. Various smoothing methods possible. Zero crossings indicate momentum shifts. Less whipsaw than raw momentum, preferred for generating actionable signals rather than pure analysis.

Syntax

var SMom(vars Series);
// or with specific periodvar SMom(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the SMom indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate SMomvars Price = series(priceClose());var value = SMom(Price);var prevvalue = SMom(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "SMom", value);

// Example trading condition
if(value > 0 && prevvalue <= 0) {
// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced SMom parameters
- Always test on historical data before using in production

1.3.22. Trix - Theme Rate of Change

Description

TEMA rate of change. Triple-smoothed ROC. Extremely smooth momentum indicator with virtually no lag compared to standard ROC. Zero line crossings generate trading signals. Reliable divergences. Combines TEMA smoothing properties with ROC momentum information. Excellent filter of false signals.

Syntax

var Trix(vars Series);
// or with specific periodvar Trix(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Trix indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Trixvars Price = series(priceClose());
var value = Trix(Price);
var prevvalue = Trix(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Trix", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Divergences with price can signal impending reversals

1.3.23. WillR - Williams' Percent Range

Description

Momentum oscillator similar to stochastic but inverted scale: -100 to 0. Calculate: ((HH - Close) / (HH - LL)) * -100. Below -80 oversold, above -20 overbought. Identical information to stochastic but different presentation. Some traders prefer inverted scale. Divergences and overbought/oversold levels used for signals.

Syntax

```
var WillR(vars Series);
// or with specific periodvar WillR(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the WillR indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate WillRvars Price = series(priceClose());var value = WillR(Price);var prevvalue = WillR(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "WillR", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Useful for identifying overbought/oversold conditions
- Divergences with price can signal impending reversals

1.4. DIGITAL FILTERS

1.4.1. AGC - Automatic Gain Control

Description

A transformation function that automatically normalizes the amplitude of a time series to maintain a constant signal level. It applies an adaptive filter that compensates for volatility variations, making the analysis of technical indicators more stable over time periods with different market conditions. Useful for strategies that require normalized signals regardless of market volatility.

Syntax

```
var AGC(vars Series);
// or with specific periodvar AGC(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the AGC indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate AGCvars Price = series(priceClose());var value = AGC(Price);var prevvalue = AGC(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "AGC", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced AGC parameters
- Always test on historical data before using in production

1.4.2. BandPass - Bandpass Filter

Description

A digital filter that isolates specific frequencies by eliminating both low- and high-frequency components. Useful for extracting market cycles of specific lengths from price time series. It allows you to focus on cyclical movements of interest by eliminating high-frequency noise and long-term trends, essential for strategies based on cyclical analysis.

Syntax

var BandPass(vars Series);
// or with specific periodvar BandPass(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the BandPass indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate BandPassvars Price = series(priceClose());var value = BandPass(Price);var prevvalue = BandPass(Price)[1]; // Previous value

```
// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "BandPass", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort) enterShort();
}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.4.3. Butterworth - Butterworth Filter

Description

Digital filter with a flat frequency response in the passband, without ripple. It offers excellent smoothing with minimal overshoot and controllable delay. It is superior to simple moving averages in reducing noise while maintaining signal integrity. Particularly effective for preprocessing data before applying machine learning algorithms or for generating clean trend-following signals.

Syntax

```
var Butterworth(vars Series);
// or with specific periodvar Butterworth(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Butterworth indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Butterworthvars Price = series(priceClose());var value = Butterworth(Price);var prevvalue = Butterworth(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Butterworth", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {
// Positive crossover - possible long signalif
```

(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible short signalif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.4.4. Decycle - Ehlers' Decycler

Description

Filter that removes cyclical components from the price, leaving only the trend. Opposite of a bandpass filter, it eliminates specific frequency fluctuations while maintaining long-term trends. Useful for identifying the true underlying trend without cyclical noise, allowing for trend-following strategies that are unaffected by regular short-term fluctuations.

Syntax

var Decycle(vars Series);
// or with specific periodvar Decycle(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Decycle indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Decyclevars Price = series(priceClose());var value = Decycle(Price);var prevvalue = Decycle(Price)[1]; // Previous value

// Log of value
if(is(LOGFILE))printf("\n%s: %.4f", "Decycle", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.4.5. Gauss - Gauss Filter

Description

This filter applies a Gaussian window for optimal smoothing with minimal overshoot and ringing. It balances smoothness and preservation of important signal features. It is superior to simple moving averages in preserving edge information while reducing noise. Latency is configurable via the standard deviation parameter. Excellent for preprocessing before applying machine learning algorithms.

Syntax

var Gauss(vars Series);
// or with specific periodvar Gauss(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Gauss indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Gaussvars Price = series(priceClose());var value = Gauss(Price);var prevvalue = Gauss(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Gauss", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced Gauss parameters
- Always test on historical data before using in production

1.4.6. HighPass - Wide Highpass Filter

Description

A filter that removes low-frequency components (long-term trends), leaving only high-frequency oscillations. It isolates noise and short-term oscillations from the trend. Useful for detrending and analyzing mean-reversion patterns without interference from the main trend. It allows you to study cyclical behavior isolated from larger directional movements.

Syntax

var HighPass(vars Series);
// or with specific periodvar HighPass(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the HighPass indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate HighPassvars Price = series(priceClose());var value = HighPass(Price);var prevvalue = HighPass(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "HighPass", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.4.7. HighPass1 - 1-pole Highpass Filter

Description

A simplified version of the single-pole highpass filter, it offers basic yet fast detrending. Rolloff is more gradual than the 2-pole filter, and the mid-range frequencies are less attenuated. Computationally efficient, it is used when only a very slow trend needs to be removed without

excessively affecting medium-period oscillations. A trade-off is between selectivity and computational simplicity.

Syntax

var HighPass1(vars Series);
// or with specific periodvar HighPass1(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the HighPass1 indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate HighPass1vars Price = series(priceClose());var value = HighPass1(Price);var prevvalue = HighPass1(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "HighPass1", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.4.8. HighPass2 - 2-pole Highpass Filter

Description

A two-pole highpass filter that offers a sharper cutoff and greater low-frequency attenuation than a single-pole filter. It removes trends more effectively while leaving the oscillations of interest intact. It is computationally more expensive but provides superior separation between trends and cyclic components. Preferred when clean isolation of oscillatory movements is required.

Syntax

var HighPass2(vars Series);
// or with specific periodvar HighPass2(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the HighPass2 indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate HighPass2vars Price = series(priceClose());var value = HighPass2(Price);var prevvalue = HighPass2(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "HighPass2", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort) enterShort();
}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.4.9. LowPass - Lowpass Filter

Description

Filter that removes high-frequency components (noise, fast oscillations), leaving only smooth trend motion. The opposite of highpass, it eliminates choppiness while maintaining a directional trend. Various designs available (Butterworth, Gaussian) with different tradeoffs between smoothness and lag. Essential preprocessing for trend-following strategies and for reducing false signals from noise.

Syntax

```
var LowPass(vars Series);
// or with specific periodvar LowPass(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the LowPass indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate LowPassvars Price = series(priceClose());var value = LowPass(Price);var prevvalue = LowPass(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "LowPass", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible long signalif
(!NumOpenLong)
enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible short signalif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.4.10. Roof - Ehlers' Roofing Filter

Description

Highpass filter followed by lowpass to extract specific frequency bands (bandpass). Removes both low-frequency trends and high-frequency noise, isolating cycles of the desired length. Superior to simple bandpass for minimizing distortion. Used in cycle-based trading to focus on dominant cycles without interference from other components.

Syntax

```
var Roof(vars Series);
// or with specific periodvar Roof(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Roof indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Roofvars Price = series(priceClose());var value = Roof(Price);var prevvalue =
Roof(Price)[1]; // Previous value

// Log of value
if(is(LOGFILE))printf("\n%s: %.4f", "Roof", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.4.11. Smooth - Ehlers' Super-Smoother

Description

Advanced two-pole Butterworth-style filter developed by John Ehlers. Superior noise reduction with minimal lag. Adjustable parameters for balance between smoothness and responsiveness. Virtually eliminates aliasing distortion common in simple moving averages. Preferred preprocessing for complex indicators and for high-frequency data cleaning.

Syntax

```
var Smooth(vars Series);
// or with specific periodvar Smooth(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Smooth indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Smoothvars Price = series(priceClose());var value = Smooth(Price);var prevvalue =
Smooth(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Smooth", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced Smooth parameters.
- Always test on historical data before using in production

1.5. 🔍 CYCLIC ANALYSIS

1.5.1. CCYI - Correlation Cycle Indicator

Description

An indicator that measures the correlation between price and cyclical components of different lengths to identify dominant market cycles. High values indicate the presence of strong and predictable cyclicality, while low values indicate a random market. Essential for strategies based on cyclical timing, it helps determine when seasonal or cyclical patterns are reliable versus when the market is dominated by non-cyclical factors.

Syntax

```
var CCYI(vars Series);
// or with specific periodvar CCYI(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the CCYI indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate CCYIvars Price = series(priceClose());var value = CCYI(Price);var prevvalue =
CCYI(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "CCYI", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced CCYI parameters
- Always test on historical data before using in production

1.5.2. CCYIR - Correlation Cycle Indicator Rate of Change

Description

Derivative of the CCYI that measures the speed of change in market cyclicality. High positive values indicate the rapid emergence of cyclical patterns (favorable for cyclical strategies), while negative values indicate a breakdown in cyclicality. Useful for adaptive strategy selection: activate cyclical strategies when the CCYIR is positive and rising, deactivate them when it becomes negative.

Syntax

```
var CCYIR(vars Series);
// or with specific periodvar CCYIR(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the CCYIR indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate CCYIRvars Price = series(priceClose());var value = CCYIR(Price);var prevvalue =
CCYIR(Price)[1]; // Previous value
```

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "CCYIR", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced CCYIR parameters
- Always test on historical data before using in production

1.5.3. CCYIState - Correlation Cycle Market State

Description

A CCYI-based market state classifier that identifies current market conditions: trending, cycling, or random. It uses thresholds based on the CCYI value and consistency to determine when cyclical patterns are safe to trade. Essential for regime-switching strategies that adapt tactics based on prevailing market behavior.

Syntax

var CCYIState(vars Series);
// or with specific periodvar CCYIState(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the CCYIState indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate CCYIStatevars Price = series(priceClose());var value = CCYIState(Price);var prevvalue = CCYIState(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "CCYIState", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();

}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.5.4. DominantPeriod - Fundamental Price Oscillation

Description

Identify the length of the dominant price cycle using spectral analysis or cycle detection algorithms. Returns the number of bars in the dominant cycle. Essential for adaptive indicators that adjust their parameters based on the current cycle pattern. Allows you to dynamically optimize moving average periods and oscillators to match market conditions.

Syntax

var DominantPeriod(vars Series);
// or with specific periodvar DominantPeriod(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the DominantPeriod indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate DominantPeriodvars Price = series(priceClose());var value = DominantPeriod(Price);var prevvalue = DominantPeriod(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))
printf("\n%s: %.4f", "DominantPeriod", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced DominantPeriod parameters
- Always test on historical data before using in production

1.5.5. DominantPhase - Fundamental Price Phase

Description

Determines the current phase of the dominant cycle (where we are in the cycle): 0° start of the upswing, 90° top, 180° start of the downswing, 270° bottom. Combined with DominantPeriod, it allows for precise timing of entries/exits synchronized with the cycle. When the phase is close to 270° (expected bottom), it's time to consider long entries, while near 90° (expected top) is time for exits or shorts.

Syntax

var DominantPhase(vars Series);
// or with specific periodvar DominantPhase(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the DominantPhase indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate DominantPhase
vars Price = series(priceClose());var value = DominantPhase(Price);var prevvalue = DominantPhase(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "DominantPhase", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced parameters.

Dominant Phase

- Always test on historical data before using in production

1.5.6. FractalDimension - Fractal Dimension

Description

Measures the geometric complexity of a time series, with values between 1 (perfect trend) and 2 (complete random walk). Values close to 1 indicate trend persistence, while values close to 2 indicate anti-persistence or mean reversion. Between 1.5 and 2 suggests favorable market conditions for mean reversion strategies, while values below 1.5 suggest trend-following strategies. Advanced market characterization tool.

Syntax

var FractalDimension(vars Series);
// or with specific periodvar FractalDimension(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the FractalDimension indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate FractalDimensionvars Price = series(priceClose());var value = FractalDimension(Price);var prevvalue = FractalDimension(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "FractalDimension", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.5.7. HTDcPeriod - Hilbert Transform Cycle Period

Description

Use the Hilbert Transform to dynamically calculate the period of the dominant price cycle. Returns the length of the prevailing price cycle in bars moment-by-moment. Adaptive indicators can use this output to automatically adjust their periods. More accurate than FFT methods for non-stationary data. Essential for building truly adaptive trading systems that self-tune.

Syntax

var HTDcPeriod(vars Series);
// or with specific periodvar HTDcPeriod(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the HTDcPeriod indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate HTDcPeriodvars Price = series(priceClose());var value = HTDcPeriod(Price);var prevvalue = HTDcPeriod(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "HTDcPeriod", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced HTDcPeriod parameters
- Always test on historical data before using in production

1.5.8. HTDcPhase - Hilbert Transform Cycle Phase

Description

Calculate the current phase of the dominant cycle (0-360°) using the Hilbert Transform. Identify where we are in the cycle: 0° upswing start, 90° peak, 180° downswing start, 270° through. Perfect

timing for cycle-synchronized entries/exits: buy near 270°, sell near 90°. More reliable than traditional cycle phase methods for non-sinusoidal prices.

Syntax

var HTDcPhase(vars Series);
// or with specific periodvar HTDcPhase(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the HTDcPhase indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate HTDcPhasevars Price = series(priceClose());var value = HTDcPhase(Price);var prevvalue = HTDcPhase(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "HTDcPhase", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort) enterShort();
}}

Usage Notes

- See the Zorro documentation for advanced HTDcPhase parameters
- Always test on historical data before using in production

1.5.9. HTPhasor - Hilbert Transform Phasor Components

Description

Returns the in-phase and quadrature components of the signal that form the phasor. These components allow for the calculation of the magnitude and phase of the analytical signal. Essential for advanced implementations of Hilbert-based indicators. The two components together provide a complete representation of the signal in time-frequency space.

Syntax

var HTPhasor(vars Series);
// or with specific periodvar HTPhasor(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the HTPhasor indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate HTPhasorvars Price = series(priceClose());var value = HTPhasor(Price);var prevvalue = HTPhasor(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "HTPhasor", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {
// Positive crossover - possible long signalif
(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible short
signalif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced HTPhasor parameters.
- Always test on historical data before using in production

1.5.10. HTSine - Hilbert Transform Sine Wave

Description

Extracts the dominant sinusoidal component using the Hilbert Transform. Generates a smoothed sine wave representing the prevailing cycle. Two outputs: a sine wave and a lead sine (advanced by 45°). Crossovers between the two generate cycle-synchronized entry/exit timing signals. Particularly useful for assets with a strong cyclic component.

Syntax

var HTSine(vars Series);
// or with specific periodvar HTSine(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the HTSine indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate HTSinevars Price = series(priceClose());var value = HTSine(Price);
var prevvalue = HTSine(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "HTSine", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced HTSine parameters
- Always test on historical data before using in production

1.5.11. HTTrendMode - Hilbert Transform Trend Indicator

Description

Binary indicator that identifies whether the market is in trending mode (1) or cycling mode (0) based on Hilbert analysis. When the output is 1, activate trend-following strategies, when 0, mean-reversion. More reliable than traditional methods in distinguishing regimes. Essential for adaptive systems that switch strategies based on market characterization.

Syntax

```
var HTTrendMode(vars Series);
// or with specific periodvar HTTrendMode(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the HTTrendMode indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate HTTrendModevars Price = series(priceClose());var value = HTTrendMode(Price);var prevvalue = HTTrendMode(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "HTTrendMode", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.5.12. Hurst - Hurst Exponent

Description

Measures long-term memory and persistence of a time series. Values: H=0.5 random walk, H>0.5 trending/persistent, H<0.5 mean-reverting. H near 0.7-0.8 indicates strong trending behavior favorable for trend-following. H near 0.3 indicates mean-reversion opportunities. Advanced market characterization tool for strategy selection and regime identification.

Syntax

```
var Hurst(vars Series);
// or with specific periodvar Hurst(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Hurst indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Hurstvars Price = series(priceClose());var value = Hurst(Price);var prevvalue = Hurst(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Hurst", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.5.13. ShannonEntropy - Randomness Metric

Description

Measure entropy (disorder/randomness) of a price series using Shannon's information theory. High entropy = high randomness = difficult to predict. Low entropy = pattern/structure = more predictability. Useful for identifying when patterns are likely to work (low entropy) vs when market is too random (high entropy). Sophisticated market characterization tool.

Syntax

var ShannonEntropy(vars Series);
// or with specific periodvar ShannonEntropy(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ShannonEntropy indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ShannonEntropyvars Price = series(priceClose());var value = ShannonEntropy(Price);var prevvalue = ShannonEntropy(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ShannonEntropy", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced ShannonEntropy parameters.
- Always test on historical data before using in production

1.5.14. ShannonGain - Expected Gain Rate

Description

Calculates expected rate of information gain from the price series. Related to entropy: how much we can expect to "learn" from future price movements. Relevant for adaptive learning systems and for assessing whether it is worth trading higher versus staying flat when expected gain is insufficient to overcome transaction costs.

Syntax

var ShannonGain(vars Series);
// or with specific periodvar ShannonGain(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ShannonGain indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ShannonGainvars Price = series(priceClose());var value = ShannonGain(Price);var prevvalue = ShannonGain(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ShannonGain", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible long signalif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible short signalif
(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced ShannonGain parameters.
- Always test on historical data before using in production

1.5.15. Spectrum - Spectral Analysis

Description

Performs frequency-domain analysis of the price series using Fast Fourier Transform. Identify dominant cycles/periodicities. Output shows amplitude for each frequency component. Reveals hidden cyclical structure. Used for identifying optimal periods for moving averages and oscillators, and for cycle-based trading strategies.

Syntax

var Spectrum(vars Series);
// or with specific periodvar Spectrum(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Spectrum indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Spectrumvars Price = series(priceClose());var value = Spectrum(Price);var prevvalue = Spectrum(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Spectrum", value);

// Example trading condition
if(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced Spectrum parameters
- Always test on historical data before using in production

1.6. 📉 SUPPORTS/RESISTANCES

1.6.1. ChandelierLong - Chandelier Exit Long

Description

Trailing stop-loss system for long positions based on the ATR. The stop is placed N times the ATR below the recent high (chandelier "hanging" from the ceiling). It moves only up, never down, protecting profits while allowing the trend to develop. The ATR multiplier determines how much room to give the trade: low values (2-3) for tight exits, high values (5-6) for riding strong trends.

Syntax

var ChandelierLong(vars Series);
// or with specific periodvar ChandelierLong(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ChandelierLong indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ChandelierLongvars Price = series(priceClose());var value = ChandelierLong(Price);var prevvalue = ChandelierLong(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ChandelierLong", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.6.2. ChandelierShort - Chandelier Exit Short

Description

Trailing stop-loss system for short positions. The stop is placed N times the ATR above the recent low. It moves only downwards, never upwards, protecting profits in downtrends. Complementary to the Chandelier Long, it allows for the implementation of symmetric trend-following strategies with risk management that adapts to volatility on both sides of the market.

Syntax

var ChandelierShort(vars Series);
// or with specific periodvar ChandelierShort(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ChandelierShort indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ChandelierShortvars Price = series(priceClose());var value = ChandelierShort(Price);var prevvalue = ChandelierShort(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ChandelierShort", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.6.3. dayClose - Day Close

Description

Returns the closing price of the last completed day. Useful in intraday strategies for referencing key levels from previous day trading or for calculating daily indicators while trading on lower timeframes. It allows for consistency across multi-timeframe analyses by incorporating daily significance levels into intraday decisions.

Syntax

var dayClose(vars Series);
// or with specific periodvar dayClose(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the dayClose indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // 1-hour timeframe

// Calculate dayClosevars Price = series(priceClose());var value = dayClose(Price);var prevvalue = dayClose(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "dayClose", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced dayClose parameters.
- Always test on historical data before using in production

1.6.4. dayHigh - Day High

Description

Returns to the high reached during the last completed day. Important resistance level for intraday strategies: breakout above the previous day's high often signals momentum continuation. Used in breakout strategies and to place stop losses above the previous day's range in short trades, providing a natural resistance level.

Syntax

var dayHigh(vars Series);
// or with specific periodvar dayHigh(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the dayHigh indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // 1-hour timeframe

// Calculate dayHighvars Price = series(priceClose());var value = dayHigh(Price);var prevvalue = dayHigh(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "dayHigh", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort) enterShort();
}}

Usage Notes

- See the Zorro documentation for advanced dayHigh parameters
- Always test on historical data before using in production

1.6.5. dayLow - Day Low

Description

Returns to the low reached during the last completed day. Important support level for intraday strategies: a break below the previous day's low indicates weakness and a possible downward continuation. Used to place stops below the previous daily range in long positions and as a target for short entries on breakdowns.

Syntax

var dayLow(vars Series);
// or with specific periodvar dayLow(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the dayLow indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // 1-hour timeframe

// Calculate dayLowvars Price = series(priceClose());var value = dayLow(Price);var prevvalue = dayLow(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "dayLow", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {
// Positive crossover - possible long signalif
(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible short signalif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced dayLow parameters
- Always test on historical data before using in production

1.6.6. dayOpen - Day Open

Description

Returns the opening price of the last completed day. The gap between the previous day's close and the next day's open is significant overnight sentiment information. The opening price is an important reference for range trading: the price often reverts to the open during the session. Also used to calculate opening range breakout strategies.

Syntax

var dayOpen(vars Series);
// or with specific periodvar dayOpen(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the dayOpen indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // 1-hour timeframe

// Calculate dayOpenvars Price = series(priceClose());var value = dayOpen(Price);
var prevvalue = dayOpen(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "dayOpen", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced dayOpen parameters.
- Always test on historical data before using in production

1.6.7. dayPivot - Day Pivot

Description

Calculate the previous day's pivot point as (High + Low + Close) / 3. The center of gravity of the previous price action, used as the equilibrium level. Price above the pivot suggests a bullish bias for the day, while below it suggests a bearish bias. Essential in intraday strategies to identify key support/resistance levels and directional bias.

Syntax

```
var dayPivot(vars Series);
// or with a specific period var dayPivot(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the dayPivot indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // 1-hour timeframe

// Calculate dayPivotvars Price = series(priceClose());var value = dayPivot(Price);var prevvalue = dayPivot(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "dayPivot", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced dayPivot parameters.
- Always test on historical data before using in production

1.6.8. DChannel - Donchian Channel

Description

A band system consisting of rolling highs and lows over N periods (typically 20). The upper band represents a breakout level for long entries, while the lower band represents a breakout level for short entries. A classic trend-following system: a breakout above the upper channel indicates strength and a possible continuation of an uptrend. The channel's width measures volatility and range conditions.

Syntax

var DChannel(vars Series);
// or with a specific period var DChannel(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the DChannel indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate DChannelvars Price = series(priceClose());var value = DChannel(Price);var prevvalue = DChannel(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "DChannel", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.6.9. FractalHigh - High Fractal Indicator

Description

Identify Williams' bullish fractal pattern: a high bar with at least two lower highs both before and after it. Mark potential resistance levels and turning points. A series of ascending fractal highs confirms an uptrend, descending fractal highs confirm a downtrend. Breakouts above the previous fractal high generate buy signals in breakout strategies.

Syntax

var FractalHigh(vars Series);
// or with specific periodvar FractalHigh(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the FractalHigh indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // 1-hour timeframe

// Calculate FractalHighvars Price = series(priceClose());var value = FractalHigh(Price);var prevvalue = FractalHigh(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "FractalHigh", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.6.10. FractalLow - Low Fractal Indicator

Description

Identify bearish fractal patterns: a low bar with at least two higher lows both before and after. Mark potential support levels and turning points. A series of descending fractal lows confirms a downtrend, while ascending fractal lows confirm an uptrend. A breakout below the previous fractal low generates a sell signal. Complementary to FractalHigh to identify complete market structure.

Syntax

var FractalLow(vars Series);
// or with specific periodvar FractalLow(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the FractalLow indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate FractalLowvars Price = series(priceClose());var value = FractalLow(Price);var prevvalue = FractalLow(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "FractalLow", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.6.11. HH - Highest High

Description

Returns the highest price reached in the last N periods. Reference key for identifying resistance levels, breakout points, and trailing stops. Rising HH confirms an uptrend; no longer updated HH may signal a loss of momentum. Used in Donchian breakouts, Chandelier stops, and countless trend-following strategies as a reference for extremes.

Syntax

var HH(vars Series);
// or with specific periodvar HH(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the HH indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate HHvars Price = series(priceClose());var value = HH(Price);var prevvalue = HH(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "HH", value);

// Example trading condition
if(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.6.12. LL - Lowest Low

Description

Returns the lowest price reached in the last N periods. Complementary to HH, it identifies support levels, potential bounce points, and trailing stop references for short positions. LL declining confirms a downtrend; LL no longer updated can signal a potential bottom. Used in breakout strategies (break below LL = short signal) and support/resistance identification.

Syntax

var LL(vars Series);
// or with specific periodvar LL(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the LL indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate LLvars Price = series(priceClose());
var value = LL(Price);
var prevvalue = LL(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "LL", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.6.13. peak - Curve Peak

Description

Identifies when a series has reached a local maximum: current value > previous and subsequent values within the specified window. True only on the exact peak bar. Essential for swing trading, structure analysis, and pattern recognition. Peak identification allows tracking swing highs for resistance levels and pattern formation.

Syntax

var peak(vars Series);
// or with specific periodvar peak(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the peak indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate peakvars Price = series(priceClose());var value = peak(Price);var prevvalue = peak(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "peak", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced peaking parameters.
- Always test on historical data before using in production

1.6.14. peakF - Curve Peak Fuzzy

Description

Fuzzy version that returns degree of peakness (0-1) instead of Boolean. Considers peak sharpness and prominence: sharp isolated peak = 1, gentle rounded peak = intermediate value. More robust to noise, allows rank ordering of peaks by significance. Useful for filtering only the most significant peaks in noisy data.

Syntax

var peakF(vars Series);
// or with specific periodvar peakF(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the peakF indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate peakFvars Price = series(priceClose());var value = peakF(Price);var prevvalue = peakF(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "peakF", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced peakF parameters.
- Always test on historical data before using in production

1.6.15. Pivot - Pivot Point

Description

Calculates pivot points and support/resistance levels derived from the previous high/low/close period. Formula: Pivot = (H+L+C)/3. Resistance levels: R1 = 2P-L, R2 = P+(HL). Support levels: S1 = 2P-H, S2 = P-(HL). Widely used in intraday trading to identify key levels where price is likely to react. Foundation of many day-trading systems.

Syntax

var Pivot(vars Series);
// or with specific periodvar Pivot(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Pivot indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Pivotvars Price = series(priceClose());var value = Pivot(Price);var prevvalue =
Pivot(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Pivot", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced Pivot parameters
- Always test on historical data before using in production

1.6.16. Resistance - Resistance Line

Description

Calculates resistance levels based on analysis of recent swing highs. Identifies price levels where selling pressure has historically emerged. Breakouts above resistance are bullish signals. The more a level is tested without a breakthrough, the more significant it becomes. Used for profit targets and to identify where long positions will encounter headwinds.

Syntax

```
var Resistance(vars Series);
// or with specific periodvar Resistance(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Resistance indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Resistancevars Price = series(priceClose());var value = Resistance(Price);var prevvalue = Resistance(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Resistance", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();
}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced Resistance parameters.
- Always test on historical data before using in production

1.6.17. Support - Support Line

Description

Calculate support levels based on analysis of recent swing lows. Identify price levels where buying pressure has historically emerged. Breakdowns below support are bearish signals. The more tested, the more significant. Used for protective stops and to identify where short positions might encounter buying interest.

Syntax

```
var Support(vars Series);
// or with specific periodvar Support(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Support indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Supportvars Price = series(priceClose());var value = Support(Price);var prevvalue = Support(Price)[1]; // Previous value

// Log of value
if(is(LOGFILE))
printf("\n%s: %.4f", "Support", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced Support parameters
- Always test on historical data before using in production

1.6.18. valley - Curve Valley

Description

Identifies when a series has reached a local minimum: current value < previous and subsequent values within the window. True only on the exact bar of the valley. Complementary to peak. Essential for swing trading, identifying support levels, and structural analysis. Valleys identify swing lows for support levels and pattern recognition.

Syntax

```
var valley(vars Series);
// or with specific periodvar valley(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the valley indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate valleyvars Price = series(priceClose());var value = valley(Price);var prevvalue =
valley(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "valley", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced valley parameters
- Always test on historical data before using in production

1.6.19. valleyF - Curve Valley Fuzzy

Description

Fuzzy version that returns degree of valley-ness (0-1). Sharp isolated valley = 1, gradual valley = intermediate value. More robust to noise, it allows ranking valleys by significance. Filters out minor valleys that could be noise, focusing on significant structural lows. Useful in noisy markets.

Syntax

```
var valleyF(vars Series);
// or with specific periodvar valleyF(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the valleyF indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate valleyFvars Price = series(priceClose());var value = valleyF(Price);var prevvalue =
valleyF(Price)[1]; // Previous value
```

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "valleyF", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced valleyF parameters.
- Always test on historical data before using in production

1.7. PATTERN RECOGNITION

1.7.1. Coral - Coral Indicator

Description

Smoothed trend-following indicator that uses a proprietary formula to reduce lag while maintaining sensitivity. Produces a colored line that changes color with trend changes. Superior to simple moving averages in trend-following with less whipsaw. Price crossovers with the Coral Line generate signals, with the distance from the line indicating trend strength.

Syntax

var Coral(vars Series);
// or with specific periodvar Coral(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Coral indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Coralvars Price = series(priceClose());var value = Coral(Price);var prevvalue = Coral(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Coral", value);

```
// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.7.2. Fisher - Fisher Transform

Description

Mathematical transformation that converts bounded series into an approximately Gaussian distribution. It emphasizes turning points, making trend reversals more evident. Unbounded outputs provide greater definition at the extremes. Zero line crossovers anticipate reversals. Particularly effective at generating early trend change signals, more responsive than traditional oscillators.

Syntax

```
var Fisher(vars Series);
// or with specific periodvar Fisher(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Fisher indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Fishervars Price = series(priceClose());var value = Fisher(Price);var prevvalue = Fisher(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Fisher", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.7.3. FisherInv - Inverse Fisher Transform

Description

An inverse transformation that converts an unbounded series into a bounded series between -1 and +1, with sigmoid curves that compress extremes. Used to normalize indicators that may have extreme values, making them smoother and more interpretable. Useful for creating bounded oscillators from unbounded indicators, facilitating the identification of overbought/oversold levels.

Syntax

var FisherInv(vars Series);
// or with specific periodvar FisherInv(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the FisherInv indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate FisherInvvars Price = series(priceClose());var value = FisherInv(Price);var prevvalue = FisherInv(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "FisherInv", value);

// Example trading condition
if(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Useful for identifying overbought/oversold conditions

1.7.4. FisherN - Fisher Transform with Normalization

Description

Combines Fisher Transform with pre-normalization of the input series. First, it normalizes the price or indicator to a standard range, then applies the Fisher Transform. It offers better performance than the classic Fisher Transform under variable volatility conditions. The resulting signals are more consistent across different market conditions and price ranges.

Syntax

var FisherN(vars Series);
// or with specific periodvar FisherN(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the FisherN indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate FisherNvars Price = series(priceClose());
var value = FisherN(Price);
var prevvalue = FisherN(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "FisherN", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced FisherN parameters
- Always test on historical data before using in production

1.7.5. frechet - Frechet Pattern Detection

Description

A pattern matching algorithm that calculates the Frechet distance between two curves to identify shape similarities. It finds historical price patterns that have similar geometry to the target pattern,

regardless of time scale or amplitude. More robust than simple pattern correlation, useful for predictive modeling based on historical similarities in price action shape.

Syntax

var frechet(vars Series);
// or with specific periodvar frechet(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the frechet indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate frechetvars Price = series(priceClose());var value = frechet(Price);var prevvalue = frechet(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "frechet", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced frechet parameters
- Always test on historical data before using in production

1.7.6. MatchIndex - Index of Best Match

Description

Finds the index in history where a target pattern best matches a template pattern. Uses correlation or distance metrics to quantify similarity. Returns an offset bar where the match is maximum. Useful for pattern-based prediction: find similar past situations and see what happened next. Foundation for analogy-based forecasting approaches.

Syntax

var MatchIndex(vars Series);
// or with specific periodvar MatchIndex(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MatchIndex indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate MatchIndexvars Price = series(priceClose());var value = MatchIndex(Price);var prevvalue = MatchIndex(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "MatchIndex", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced MatchIndex parameters.
- Always test on historical data before using in production

1.7.7. ZigZag - ZigZag Indicator

Description

Filters out movements smaller than a specified percentage threshold, connecting only significant swing highs and lows. Produces clean trendline patterns by removing noise. Useful for identifying major trends and structural levels. Visual aid rather than direct trading signal. Non-predictive (does not know if the current swing has completed until the next reversal).

Syntax

var ZigZag(vars Series);
// or with specific periodvar ZigZag(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ZigZag indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ZigZagvars Price = series(priceClose());var value = ZigZag(Price);var prevvalue = ZigZag(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ZigZag", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.8. 📊 STATISTICS AND ANALYSIS

1.8.1. Amplitude - Amplitude of Series

Description

Calculates the maximum amplitude of a time series over a specified period. It measures the difference between the highest and lowest values of the series, useful for assessing volatility and range of movement. It can be applied to any series (prices, indicators, oscillators) to quantify the intensity of fluctuations and identify periods of high or low volatility.

Syntax

var Amplitude(vars Series);
// or with specific periodvar Amplitude(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Amplitude indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Amplitude vars Price = series(priceClose()); var value = Amplitude(Price); var prevvalue = Amplitude(Price)[1]; // Previous value

// Log of value if(is(LOGFILE)) printf("\n%s: %.4f", "Amplitude", value);

// Example trading condition if(value > 0 && prevvalue <= 0) { // Positive crossover - possible signal long if(!NumOpenLong)
enterLong();
}

if(value < 0 && prevvalue >= 0) { // Negative crossover - possible signal short
if(!NumOpenShort)enterShort(); }}

Usage Notes

- See the Zorro documentation for advanced Amplitude parameters
- Always test on historical data before using in production

1.8.2. ATR - Average True Range (Original)

Description

Measures volatility by averaging the true range over N periods (typically 14). The true range is the maximum of: (High-Low), abs(previous High-Close), abs(previous Low-Close). Expressed in price units, high values indicate high volatility. Essential for sizing positions and determining stop losses that adapt to market volatility.

Syntax

var ATR(vars Series);
// or with specific periodvar ATR(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ATR indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ATRvars Price = series(priceClose());var value = ATR(Price);var prevvalue = ATR(Price)[1];
// Previous value

// Log of value
if(is(LOGFILE))printf("\n%s: %.4f", "ATR", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced ATR parameters
- Always test on historical data before using in production

1.8.3. ATRS - Average True Range (Simple MA)

Description

A variation of the ATR that uses a simple moving average instead of an exponential one for its calculation. It is less reactive to sudden changes in volatility but more stable. It is preferable when you want a volatility measure less sensitive to temporary spikes, useful for long-term strategies or when you want to avoid frequent stop-loss adjustments.

Syntax

var ATRS(vars Series);
// or with specific periodvar ATRS(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ATRS indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ATRSvars Price = series(priceClose());var value = ATRS(Price);var prevvalue =
ATRS(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ATRS", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced ATRS parameters
- Always test on historical data before using in production

1.8.4. Beta - Beta Value

Description

A coefficient that measures an asset's sensitivity to a benchmark (typically a market index). Beta = 1 means aligned with the market, > 1 means more volatile than the market, < 1 means less volatile, and negatively opposite movement. Calculated using linear regression of returns. Essential for portfolio management, hedging, and systematic risk assessment.

Syntax

```
var Beta(vars Series);
// or with specific periodvar Beta(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Beta indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Betavars Price = series(priceClose());var value = Beta(Price);var prevvalue =
Beta(Price)[1]; // Previous value
```

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Beta", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced Beta parameters
- Always test on historical data before using in production

1.8.5. Correlation - Pearson Correlation Coefficient

Description

Calculates the Pearson correlation coefficient between two time series, measuring how much they move together. Values range from -1 (perfect negative correlation) to +1 (perfect positive correlation), with 0 indicating no linear correlation. Essential for pair trading, portfolio diversification, and spread analysis. It allows you to identify when historical correlations break down, creating opportunities.

Syntax

var Correlation(vars Series);
// or with specific periodvar Correlation(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Correlation indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Correlationvars Price = series(priceClose());var value = Correlation(Price);var prevvalue = Correlation(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Correlation", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal

longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Correlation parameters
- Always test on historical data before using in production

1.8.6. Covariance - Covariance Coefficient

Description

Calculates the covariance between two time series, measuring how much they vary together. Similar to correlation but not normalized, it depends on the units of the variables. A positive correlation indicates a tendency to move in the same direction, while a negative correlation indicates a tendency to move in opposite directions. Used in portfolio optimization (Markowitz) to calculate portfolio risk and in pair selection for spread trading strategies.

Syntax

var Covariance(vars Series);
// or with specific periodvar Covariance(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Covariance indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Covariance vars Price = series(priceClose());var value = Covariance(Price);var prevvalue = Covariance(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Covariance", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced Covariance parameters
- Always test on historical data before using in production

1.8.7. MaxVal - Highest Value

Description

Returns the maximum value (not necessarily price) found in a series in the last N periods. Applicable to any data series: indicators, volume, oscillators. Used for normalization, identifying extremes, and comparing with current values to determine if we are close to historical maxima.

Syntax

var MaxVal(vars Series);
// or with specific periodvar MaxVal(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MaxVal indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate MaxValvars Price = series(priceClose());var value = MaxVal(Price);var prevvalue = MaxVal(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))
printf("\n%s: %.4f", "MaxVal", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Incorporates volume information for more comprehensive analysis

1.8.8. MaxIndex - Index of Highest Value

Description

Returns the index (bars ago) where the maximum value occurred in the last N periods. Combined with MaxVal, it not only indicates the maximum value but also when it occurred. Useful for timing: if MaxIndex is recent, it suggests a possible imminent reversal; if old, it suggests continued sustained momentum.

Syntax

var MaxIndex(vars Series);
// or with specific periodvar MaxIndex(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MaxIndex indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate MaxIndex
vars Price = series(priceClose());
var value = MaxIndex(Price);var prevvalue = MaxIndex(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "MaxIndex", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced MaxIndex parameters.
- Always test on historical data before using in production

1.8.9. Median - Median Filter

Description

Calculates the median (central value) of a series over N periods. More robust to outliers than the mean: single spikes do not affect the output. Excellent noise reduction while maintaining sharp edges. Preferred when data contains occasional outliers that would distort the mean. Produces stepped output when data has plateaus.

Syntax

var Median(vars Series);
// or with specific periodvar Median(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Median indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Medianvars Price = series(priceClose());var value = Median(Price);var prevvalue = Median(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Median", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced Median parameters
- Always test on historical data before using in production

1.8.10. MinVal - Lowest Value

Description

Returns the minimum value found in a series in the last N periods. Complementary to MaxVal, applicable to any series. Used to identify support levels, floor values, and to calculate ranges. Combined with MaxVal, it determines the amplitude of the recent movement in the series.

Syntax

var MinVal(vars Series);
// or with specific periodvar MinVal(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MinVal indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate MinValvars Price = series(priceClose());var value = MinVal(Price);var prevvalue = MinVal(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "MinVal", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See Zorro documentation for advanced MinVal parameters
- Always test on historical data before using in production

1.8.11. MinIndex - Index of Lowest Value

Description

Returns the index (bars ago) of the minimum value in the last N periods. Timing information: Recent MinIndex suggests a possible imminent bounce, old suggests sustained bearish pressure. Combined with MinVal, it provides complete information on when and where the minimum occurred.

Syntax

var MinIndex(vars Series);
// or with specific periodvar MinIndex(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MinIndex indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate MinIndexvars Price = series(priceClose());var value = MinIndex(Price);var prevvalue = MinIndex(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "MinIndex", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced MinIndex parameters.
- Always test on historical data before using in production

1.8.12. MinMax - Lowest and Highest Values

Description

Returns both the minimum and maximum of a series in a single function call. More efficient than calling MinVal and MaxVal separately. Allows you to calculate the range (Max-Min) and determine the current position within the range in a single operation. Used when you need to know both extremes.

Syntax

var MinMax(vars Series);
// or with specific periodvar MinMax(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MinMax indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate MinMaxvars Price = series(priceClose());var value = MinMax(Price);var prevvalue = MinMax(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "MinMax", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced MinMax parameters
- Always test on historical data before using in production

1.8.13. MinMaxIndex - Indexes of Min and Max

Description

Returns the minimum and maximum indices in single calls. Provides timing information for both extremes. Allows you to analyze swing structure: if MinIndex > MaxIndex, then the last swing was up, vice versa, down. Essential for swing-based analysis and structure identification.

Syntax

```
var MinMaxIndex(vars Series);
// or with specific periodvar MinMaxIndex(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the MinMaxIndex indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour
```

```
// Calculate MinMaxIndexvars Price = series(priceClose());var value = MinMaxIndex(Price);var prevvalue = MinMaxIndex(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "MinMaxIndex", value);

// Example trading condition
if(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced MinMaxIndex parameters.
- Always test on historical data before using in production

1.8.14. Mode - Most Frequent Value

Description

Returns the most common value in a series over the last N periods. Different from the mean and median, it identifies the value that appears most frequently. Useful for discretized data or when identifying the most popular or frequently traded price level in a period. It can reveal levels of consolidation and congestion.

Syntax

```
var Mode(vars Series);
// or with specific periodvar Mode(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Mode indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Mode
vars Price = series(priceClose());var value = Mode(Price);var prevvalue = Mode(Price)[1]; // Previous value
```

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Mode", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced Mode parameters
- Always test on historical data before using in production

1.8.15. Moment - Mean, Variance, Skew, Kurtosis

Description

Calculates all four statistical moments of a single call distribution: mean (center), variance (spread), skewness (skewness), and kurtosis (tailedness). Complete statistical characterization of the series. Mean indicates average level, variance indicates volatility, skew indicates directional bias, and kurtosis indicates the probability of extreme events. Essential for distributional analysis and risk assessment.

Syntax

var Moment(vars Series);
// or with specific periodvar Moment(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Moment indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Momentvars Price = series(priceClose());var value = Moment(Price);var prevvalue = Moment(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Moment", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal

longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced Moment parameters
- Always test on historical data before using in production

1.8.16. NATR - Normalized Average True Range

Description

ATR normalized from the current price, expressed as a percentage instead of dollars. It allows you to compare volatility across different assets and timeframes. NATR 2% means typical daily volatility is 2% of the price. Useful for cross-asset position sizing and to identify when an asset is unusually volatile compared to its historical norm.

Syntax

var NATR(vars Series);
// or with a specific period var NATR(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the NATR indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate NATRvars Price = series(priceClose());var value = NATR(Price);var prevvalue = NATR(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "NATR", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced NATR parameters
- Always test on historical data before using in production

1.8.17. NumInRange - Count Ranges in Interval

Description

Counts how many values in a series fall within a specified range [low, high] over the last N periods. Useful for density analysis: identifying where price has spent most of its time. High counts at certain levels indicate support/resistance importance. Can be used to construct volume profiles or price distribution histograms algorithmically.

Syntax

var NumInRange(vars Series);
// or with specific periodvar NumInRange(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the NumInRange indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate NumInRangevars Price = series(priceClose());var value = NumInRange(Price);var prevvalue = NumInRange(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "NumInRange", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort) enterShort();
}}

Usage Notes

- Incorporates volume information for more comprehensive analysis

1.8.18. Percentile - Percentile

Description

Calculates the percentile of a value in a distribution. Example: 90th percentile means 90% of the values are below. Useful for determining if the current value is extreme: above the 95th percentile indicates overbought conditions, below the 5th percentile indicates oversold. Better alternatives to fixed thresholds that do not adapt to changing volatility and range.

Syntax

var Percentile(vars Series);
// or with specific periodvar Percentile(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Percentile indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Percentilevars Price = series(priceClose());var value = Percentile(Price);var prevvalue = Percentile(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Percentile", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible long signalif
(!NumOpenLong)
enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible short signalif(!NumOpenShort)enterShort();}}

Usage Notes

- Useful for identifying overbought/oversold conditions

1.8.19. PercentRank - Percent Rank

Description

Returns the percentage position of the current value in the recent distribution (0-100). Similar to percentile but calculated differently. 100 means the high of the period, 0 means the low, and 50 means the middle. Self-normalizing indicator that automatically adapts to changing ranges, useful for overbought/oversold identification.

Syntax

var PercentRank(vars Series);
// or with specific periodvar PercentRank(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the PercentRank indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate PercentRankvars Price = series(priceClose());var value = PercentRank(Price);var prevvalue = PercentRank(Price)[1]; // Previous value

// Log of value
if(is(LOGFILE))printf("\n%s: %.4f", "PercentRank", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Useful for identifying overbought/oversold conditions

1.8.20. ProfitFactor - Ratio of Positive to Negative Returns

Description

Calculates gross profit divided by gross loss on a series of returns. Values >1 indicate more profit than loss, >2 is considered good, and >3 is excellent. This performance measure penalizes loss volatility. More robust than average return because it considers the full distribution. Used to evaluate signal robustness and optimize parameters.

Syntax

var ProfitFactor(vars Series);
// or with specific periodvar ProfitFactor(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ProfitFactor indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ProfitFactorvars Price = series(priceClose());var value = ProfitFactor(Price);var prevvalue = ProfitFactor(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ProfitFactor", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced ProfitFactor parameters.
- Always test on historical data before using in production

1.8.21. R2 - Determination Coefficient

Description

Measures the goodness of fit of a regression, from 0 to 1. R2 close to 1 indicates the data fits the linear trend perfectly, while an R2 close to 0 indicates no linear trend. Used to identify when the price is in a strong trend (high R2, suitable for trend-following) versus choppy (low R2, suitable for mean reversion). An essential filter for trend-quality assessment.

Syntax

var R2(vars Series);
// or with specific periodvar R2(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the R2 indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate R2vars Price = series(priceClose());var value = R2(Price);var prevvalue = R2(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "R2", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.8.22. SemiMoment - Mean, Downside Variance, Skew, Kurtosis

Description

Calculates statistical moments by focusing only on downside deviation. Downside variance only considers returns below mean. More relevant for risk assessment than standard variance because it distinguishes upside volatility (good) from downside volatility (bad). Essential for the Sortino ratio and downside risk-adjusted performance metrics.

Syntax

var SemiMoment(vars Series);
// or with specific periodvar SemiMoment(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the SemiMoment indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate SemiMomentvars Price = series(priceClose());var value = SemiMoment(Price);var prevvalue = SemiMoment(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "SemiMoment", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced SemiMoment parameters.
- Always test on historical data before using in production

1.8.23. Sharpe - Sharpe Ratio

Description

Calculate Sharpe ratio: (Mean Return - Risk-Free Rate) / Standard Deviation. Measures excess return per unit of total risk. Values >1 are good, >2 are very good, and >3 are excellent. Most common risk-adjusted performance metric. Penalizes volatility even if it's positive. Higher Sharpe indicates better risk-adjusted returns.

Syntax

var Sharpe(vars Series);
// or with specific periodvar Sharpe(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Sharpe ratio for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Sharpevars Price = series(priceClose());var value = Sharpe(Price);var prevvalue =
Sharpe(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Sharpe", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)
enterShort();
}}
```

Usage Notes

- See the Zorro documentation for advanced Sharpe parameters
- Always test on historical data before using in production

1.8.24. Sortino - Sortino Ratio

Description

Similar to the Sharpe ratio, but uses downside deviation instead of total standard deviation in the denominator. It penalizes only downside volatility, not upside. (Mean Return - Target) / Downside Deviation. More appropriate performance metric because upside volatility is desirable. Values >2 are considered good. Better than Sharpe for assessing strategies with asymmetric returns.

Syntax

```
var Sortino(vars Series);
// or with a specific period var Sortino(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Sortino indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Sortinovars Price = series(priceClose());var value = Sortino(Price);var prevvalue =
```

```
Sortino(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Sortino", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {
// Positive crossover - possible long signalif
(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible short
signalif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced Sortino parameters.
- Always test on historical data before using in production

1.8.25. Spearman - Spearman's Rank Correlation

Description

Calculates correlation using ranks instead of raw values. Non-parametric version of Pearson correlation. Captures monotonic relationships even if they are non-linear. More robust to outliers and non-normal distributions. Values -1 to +1 like Pearson. Useful for detecting relationships in noisy data or when the distribution is non-normal.

Syntax

```
var Spearman(vars Series);
// or with specific periodvar Spearman(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Spearman indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Spearmanvars Price = series(priceClose());var value = Spearman(Price);
var prevvalue = Spearman(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Spearman", value);
```

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced Spearman parameters.
- Always test on historical data before using in production

1.8.26. StdDev - Standard Deviation

Description

Measurement of dispersion of returns around the mean. High StdDev = high volatility/risk. Foundation for Bollinger Bands, Sharpe ratios, and many risk metrics. Assume normal distribution. More StdDev means more uncertainty and potential for large moves both directions. Essential component in position sizing and risk management.

Syntax

var StdDev(vars Series);
// or with specific periodvar StdDev(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the StdDev indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate StdDevvars Price = series(priceClose());var value = StdDev(Price);var prevvalue = StdDev(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "StdDev", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced StdDev parameters
- Always test on historical data before using in production

1.8.27. TrueRange - True Range

Description

Maximum of: (High-Low), abs(High-ClosePrev), abs(Low-ClosePrev). Captures full range of movement including gaps. Foundation for ATR calculation. Measure actual volatility experienced per bar. Large true range indicates high volatility; important for position sizing, stop placement, and volatility-adjusted indicators.

Syntax

var TrueRange(vars Series);
// or with specific periodvar TrueRange(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the TrueRange indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate TrueRangevars Price = series(priceClose());var value = TrueRange(Price);var prevvalue = TrueRange(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "TrueRange", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced TrueRange parameters
- Always test on historical data before using in production

1.8.28. Variance - Variance

Description

Calculate variance of returns: average of squared deviations from the mean. Foundation for standard deviation (sqrt of variance). Measures dispersion. High variance = high unpredictability and risk. Used in portfolio theory (modern portfolio theory), option pricing, and all risk calculations. Assume normal distribution for complete interpretation.

Syntax

var Variance(vars Series);
// or with specific periodvar Variance(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Variance indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Variance vars Price = series(priceClose());var value = Variance(Price);var prevvalue = Variance(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Variance", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced Variance parameters
- Always test on historical data before using in production

1.8.29. Volatility - Annualized Volatility

Description

Calculate annualized volatility from returns: standard deviation * sqrt(252 for daily, 52 for weekly). Expresses risk in annualized terms for comparability. Essential input for option pricing, position

sizing, risk management. Historical volatility used as a proxy for future volatility. Critical parameter for Sharpe, Sortino, and risk-adjusted metrics.

Syntax

var Volatility(vars Series);
// or with specific periodvar Volatility(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Volatility indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Volatility vars Price = series(priceClose()); var value = Volatility(Price); var prevvalue = Volatility(Price)[1]; // Previous value

// Log of value if(is(LOGFILE)) printf("\n%s: %.4f", "Volatility", value);

// Example trading condition if(value > 0 && prevvalue <= 0) { // Positive crossover - possible signal long if(!NumOpenLong)enterLong();
}

if(value < 0 && prevvalue >= 0) { // Negative crossover - possible signal short
if(!NumOpenShort)enterShort(); }}

Usage Notes

- See the Zorro documentation for advanced Volatility parameters.
- Always test on historical data before using in production

1.8.30. VolatilityC - Chaikin Volatility Indicator

Description

Measure volatility as percentage change in difference between high and low. Monitors trading range expansion/contraction. Increasing Chaikin volatility can precede large moves. Decreasing indicates consolidation that often precedes breakouts. Different approach to volatility measurement than standard deviation.

Syntax

var VolatilityC(vars Series);
// or with specific periodvar VolatilityC(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the VolatilityC indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate VolatilityCvars Price = series(priceClose());var value = VolatilityC(Price);var prevvalue = VolatilityC(Price)[1]; // Previous value

// Log of value
if(is(LOGFILE))
printf("\n%s: %.4f", "VolatilityC", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced VolatilityC parameters.
- Always test on historical data before using in production

1.8.31. VolatilityMM - Min/Max Volatility

Description

Calculate volatility as (Max-Min)/Mean over specified period. Normalize range by average level. Alternative volatility measures that directly capture price range dynamics. High MM volatility indicates large swings relative to price level. Useful when you want range-based rather than deviation-based volatility measure.

Syntax

var VolatilityMM(vars Series);
// or with specific periodvar VolatilityMM(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the VolatilityMM indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // 1-hour timeframe

// Calculate VolatilityMMvars Price = series(priceClose());var value = VolatilityMM(Price);var prevvalue = VolatilityMM(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "VolatilityMM", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced VolatilityMM parameters.
- Always test on historical data before using in production

1.8.32. VolatilityOV - Empirical Volatility

Description

Calculate realized volatility using actual high-low-close data with Garman-Klass or Parkinson formula. More efficient estimate of true volatility than using only closes. Incorporates intrabar information. Better estimate when you have access to OHLC data rather than just settlement prices. Preferred by option traders.

Syntax

```
var VolatilityOV(vars Series);
// or with specific periodvar VolatilityOV(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the VolatilityOV indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate VolatilityOVvars Price = series(priceClose());var value = VolatilityOV(Price);var prevvalue = VolatilityOV(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "VolatilityOV", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced VolatilityOV parameters.
- Always test on historical data before using in production

1.9. 🔗 UTILITY FUNCTIONS AND TRANSFORMATIONS

1.9.1. Concave - Curve Concavity

Description

Measures the concavity (curvature) of a time series, identifying whether the curve is accelerating upward (convex) or downward (concave). Calculated from the second derivative of the series. Useful for identifying changes in momentum: when it changes from concave to convex, it suggests the end of a deceleration and the beginning of a bullish acceleration; vice versa for the opposite transition.

Syntax

var Concave(vars Series);
// or with specific periodvar Concave(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Concave indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Concavers Price = series(priceClose());var value = Concave(Price);var prevvalue = Concave(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Concave", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced Concave parameters.
- Always test on historical data before using in production

1.9.2. CrossOver - Curve Cross Over

Description

Identifies when a time series (e.g., fast MA) crosses upward through another time series (e.g., slow MA). Returns true only on the exact bar of the crossover. Essential for moving average crossover systems and momentum strategies. Can be applied to any combination of time series: price vs. indicator, indicator vs. threshold level, or two indicators together.

Syntax

var CrossOver(vars Series);
// or with specific periodvar CrossOver(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the CrossOver indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate CrossOvervars Price = series(priceClose());var value = CrossOver(Price);var prevvalue = CrossOver(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "CrossOver", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced parameters.

CrossOver

- Always test on historical data before using in production

1.9.3. CrossOverF - Fuzzy Cross Over

Description

A fuzzy logic version of crossover that doesn't require a net crossover but considers "quasi-crossovers" within a tolerance range. It reduces false signals from whipsaw by returning degrees of membership (0-1) instead of Boolean values. Useful in choppy markets where net crossovers are rare and frequently reversed, it allows for graduated entries based on conviction level.

Syntax

var CrossOverF(vars Series);
// or with specific periodvar CrossOverF(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the CrossOverF indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate CrossOverFvars Price = series(priceClose());var value = CrossOverF(Price);var prevvalue = CrossOverF(Price)[1]; // Previous value

// Log of value
if(is(LOGFILE))printf("\n%s: %.4f", "CrossOverF", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced CrossOverF parameters.
- Always test on historical data before using in production

1.9.4. CrossUnder - Curves Cross Under

Description

Identifies when a time series crosses downward through another time series. Complementary to CrossOver, it returns true only on the exact bar of the downward crossing. Used as an exit signal in long systems or an entry signal in short systems. Essential in all systems based on indicator interactions or comparisons with threshold levels.

Syntax

var CrossUnder(vars Series);
// or with specific periodvar CrossUnder(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the CrossUnder indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate CrossUndervars Price = series(priceClose());var value = CrossUnder(Price);var prevvalue = CrossUnder(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "CrossUnder", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced CrossUnder parameters.
- Always test on historical data before using in production

1.9.5. CrossUnderF - Fuzzy Cross Under

Description

Fuzzy logic version of the cross under with the same logic as the CrossOverF but for downward crossings. It allows for the identification of "near cross unders" with degrees of membership between 0 and 1. It reduces whipsaw in sideways markets and allows for the implementation of graduated exit strategies instead of binary ones, modulating the size of the exit based on the strength of the signal.

Syntax

var CrossUnderF(vars Series);
// or with a specific period var CrossUnderF(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the CrossUnderF indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate CrossUnderFvars Price = series(priceClose());var value = CrossUnderF(Price);var prevvalue = CrossUnderF(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "CrossUnderF", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced CrossUnderF parameters.
- Always test on historical data before using in production

1.9.6. Divergence - Price/Oscillator Peak Line Divergence

Description

Identify divergences between price and the momentum oscillator: price makes a higher high but the oscillator lowers high (bearish divergence), or price makes a lower low but the oscillator higher low (bullish divergence). Divergences are among the most reliable signals of an impending trend reversal, indicating a loss of momentum. Essential in reversion strategies and entry/exit timing.

Syntax

var Divergence(vars Series);
// or with specific periodvar Divergence(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Divergence indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Divergencevars Price = series(priceClose());var value = Divergence(Price);var prevvalue = Divergence(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Divergence", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

- Divergences with price can signal impending reversals

1.9.7. falling - Curve Falling

Description

Returns true if the time series is falling (current value < previous value). A simple yet powerful function for identifying negative momentum. It can be applied to prices, indicators, and volume. Useful in rule-based systems for entry/exit conditions based on falling prices, momentum indicators, or volume. Combined with other signals, it forms sophisticated filters.

Syntax

var falling(vars Series);
// or with specific periodvar falling(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the falling indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate fallingvars Price = series(priceClose());var value = falling(Price);var prevvalue = falling(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "falling", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible long signalif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible short signalif
(!NumOpenShort)enterShort();}}

Usage Notes

- Incorporates volume information for more comprehensive analysis

1.9.8. fallingF - Curve Falling Fuzzy

Description

Fuzzy logic version of falling that returns degree of fallingness (0-1) instead of Boolean. Considers slope and consistency of the decline. Minor slopes or intermittent declines return values < 1. Allows gradual quantification of the falling condition, useful for variable position sizing and partial entries/exits based on the signal's conviction level.

Syntax

var fallingF(vars Series);
// or with specific periodvar fallingF(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the fallingF indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate fallingFvars Price = series(priceClose());var value = fallingF(Price);var prevvalue = fallingF(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "fallingF", value);

// Example trading condition
if(value > 0 && prevvalue <= 0) {
// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced fallingF parameters.
- Always test on historical data before using in production

1.9.9. findIdx - Find Element

Description

Searches for a specific element in an array and returns its index. If not found, it returns -1. Useful for lookup operations on data arrays, identifying specific occurrences, and implementing algorithms that

require search operations. It can be used to find specific pattern values, threshold crossings, or matching conditions in time series.

Syntax

var findIdx(vars Series);
// or with specific periodvar findIdx(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the findIdx indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate findIdxvars Price = series(priceClose());
var value = findIdx(Price);
var prevvalue = findIdx(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "findIdx", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced findIdx parameters.
- Always test on historical data before using in production

1.9.10. line - Line Position at Given Bar

Description

Calculates the value of a linear regression line at a specific bar. Useful for extending trend lines beyond fitting data or calculating projected support/resistance levels. Combined with slope, it allows you to project the expected trend. Foundation for many pattern recognition algorithms and trendline-based strategies.

Syntax

var line(vars Series);
// or with specific periodvar line(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the line indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate linevars Price = series(priceClose());var value = line(Price);var prevvalue = line(Price)[1];
// Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "line", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.9.11. LinearRegAngle - Linear Regression Angle

Description

Returns the angle (in degrees) of the regression line with respect to the horizontal axis. Positive angles indicate an uptrend, negative ones indicate a downtrend. The magnitude of the angle quantifies the steepness of the trend. Extreme angles (>45° or <-45°) indicate an unsustainable trend, close to 0, a weak trend, or sideways. Useful metric for classifying trend strength and for filtering trades based on trend quality.

Syntax

var LinearRegAngle(varsSeries);
// or with specific periodvar LinearRegAngle(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the LinearRegAngle indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate LinearRegAnglevars Price = series(priceClose());var value = LinearRegAngle(Price);var prevvalue = LinearRegAngle(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "LinearRegAngle", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.9.12. LinearRegIntercept - Linear Regression Intercept

Description

Returns the Y value where the regression line crosses the Y axis (when X=0). Intercept combined with slope completely defines the regression line. Used to calculate projected values of the regression line at any point. Essential component when implementing strategies based on channels or deviations from the regression line.

Syntax

var LinearRegIntercept(vars Series);
// or with specific periodvar LinearRegIntercept(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the LinearRegIntercept indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate LinearRegInterceptvars Price = series(priceClose());var value = LinearRegIntercept(Price);var prevvalue = LinearRegIntercept(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "LinearRegIntercept", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced LinearRegIntercept parameters.
- Always test on historical data before using in production

1.9.13. LinearRegSlope - Linear Regression Slope

Description

Returns the slope of the regression line (change in Y per unit change in X). Positive slope indicates uptrend, negative downtrend. Magnitude quantifies the speed of the trend: high values indicate rapid movement. Slope changing from positive to negative or vice versa signals a trend reversal. Foundation for slope-based indicators and for calculating price movement velocity.

Syntax

var LinearRegSlope(vars Series);
// or with specific periodvar LinearRegSlope(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the LinearRegSlope indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate LinearRegSlopevars Price = series(priceClose());var value = LinearRegSlope(Price);var prevvalue = LinearRegSlope(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "LinearRegSlope", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.9.14. Normalize - Normalize to -1 .. +1

Description

Transforms a series in the range -1 to +1 using min-max scaling over a specified period. Facilitates comparisons between different indicators and assets. Normalized values close to +1 indicate near the range's maximum, while -1 indicates near the range's minimum. Essential preprocessing for neural networks and when combining indicators with different scales.

Syntax

```
var Normalize(vars Series);
// or with specific periodvar Normalize(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Normalize indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Normalizevars Price = series(priceClose());var value = Normalize(Price);var prevvalue = Normalize(Price)[1]; // Previous value
```

```
// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Normalize", value);

// Example trading condition
if(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced Normalize parameters.
- Always test on historical data before using in production

1.9.15. NumDn - Count of Falling Elements

Description

Counts how many consecutive or total bars in the last N periods have moved downward. High counts indicate sustained bearish pressure. May indicate oversold conditions when NumDn is extremely high. Used in contrarian strategies: look for reversals up when NumDn reaches extremes.

Syntax

```
var NumDn(vars Series);
// or with specific periodvar NumDn(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the NumDn indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate NumDn
vars Price = series(priceClose());var value = NumDn(Price);var prevvalue = NumDn(Price)[1]; //
Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "NumDn", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}
```

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Useful for identifying overbought/oversold conditions

1.9.16. NumRiseFall - Length of Streak

Description

Returns the length of the current streak of rising or falling bars. Positive values for upstreaks, negative for downstreaks. Long streaks indicate strong momentum but also possible overextension. Extremely long streaks often precede reversals. Useful for identifying exhaustion points and mean-reversion opportunities.

Syntax

var NumRiseFall(vars Series);
// or with specific periodvar NumRiseFall(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the NumRiseFall indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate NumRiseFallvars Price = series(priceClose());var value = NumRiseFall(Price);var prevvalue = NumRiseFall(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "NumRiseFall", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced NumRiseFall parameters.

- Always test on historical data before using in production

1.9.17. NumUp - Count of Rising Elements

Description

Counts how many consecutive or total bars in the last N periods have moved upward. Complementary to NumDn. High counts indicate sustained bullish pressure. It can signal overbought when NumUp is extremely high. Divergences between NumUp and price can herald reversals.

Syntax

var NumUp(vars Series);
// or with specific periodvar NumUp(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the NumUp indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate NumUpvars Price = series(priceClose());var value = NumUp(Price);var prevvalue = NumUp(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "NumUp", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Useful for identifying overbought/oversold conditions
- Divergences with price can signal impending reversals

1.9.18. NumWhiteBlack - Difference White and Black Candles

Description

Counts the difference between white (close > open) and black (close < open) candles over N periods. Positive values indicate a preponderance of bullish candles, while negative values indicate a preponderance of bearish candles. Measures sentiment and buying vs. selling pressure. Series declining while price rising indicates a bearish warning divergence.

Syntax

var NumWhiteBlack(vars Series);
// or with specific periodvar NumWhiteBlack(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the NumWhiteBlack indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate NumWhiteBlackvars Price = series(priceClose());var value = NumWhiteBlack(Price);var prevvalue = NumWhiteBlack(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "NumWhiteBlack", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Divergences with price can signal impending reversals

1.9.19. polyfit - Polynomial Regression

Description

Calculates polynomial regression of a specified order (quadratic, cubic, etc.) on data. Higher-order polynomials fit nonlinear curves. The output is the coefficients of the polynomial that best fits the data. Useful for modeling accelerating/decelerating trends (quadratic) and complex curves (cubic+). More flexible than linear regression for nonlinear patterns.

Syntax

var polyfit(vars Series);
// or with specific periodvar polyfit(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the polyfit indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate polyfitvars Price = series(priceClose());var value = polyfit(Price);var prevvalue = polyfit(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "polyfit", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.9.20. polynom - Regression Polynomial

Description

Evaluates the polynomial obtained from polyfit at a specified point. Combined with polyfit, it allows you to project fitted curves beyond the data points. Used for forecasting based on polynomial trend assumptions or to calculate expected values according to the polynomial model. Essential for implementing polynomial-based technical analysis.

Syntax

var polynom(vars Series);
// or with specific periodvar polynom(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the polynom indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate polynomvars Price = series(priceClose());var value = polynom(Price);var prevvalue = polynom(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "polynom", value);

// Example trading condition
if(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.9.21. predict - Curve Peak/Crossover Prediction

Description

Attempts to predict bars until the next peak or crossover using extrapolation of the current trend. Analyzes the series' velocity and acceleration to project when it will reach an extreme or through. Probabilistic, non-deterministic forecast. Used for anticipatory positioning or to set profit targets based on expected timing of reversals.

Syntax

var predict(vars Series);
// or with specific periodvar predict(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the predict indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate predict
vars Price = series(priceClose());
var value = predict(Price);var prevvalue = predict(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "predict", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends

1.9.22. predictMove - Predict Price Move by Statistics

Description

Uses statistical analysis of past moves similar to current conditions to predict the next move. It searches for historical patterns that match the current setup and calculates the distribution of results. Output: expected move and confidence interval. More sophisticated than simple pattern matching, it incorporates a probabilistic framework.

Syntax

```
var predictMove(vars Series);
// or with specific periodvar predictMove(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the predictMove indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate predictMovevars Price = series(priceClose());var value = predictMove(Price);var prevvalue = predictMove(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "predictMove", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced predictMove parameters.
- Always test on historical data before using in production

1.9.23. predictSeason - Predict Price Move by Seasonal Analysis

Description

Analyze historical seasonality (day-of-week, month-of-year, etc.) to predict the direction and magnitude of the next move. Identify whether certain periods have consistent bias. Example: equity tends to rise in the first few days of the month, then falls before the weekend. Useful for adding seasonal edge to strategies or for timing entries/exits based on seasonal probabilities.

Syntax

```
var predictSeason(vars Series);
// or with specific periodvar predictSeason(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the predictSeason indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate predictSeasonvars Price = series(priceClose());var value = predictSeason(Price);var prevvalue = predictSeason(Price)[1]; // Previous value
```

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "predictSeason", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced predictSeason parameters.
- Always test on historical data before using in production

1.9.24. QLSMA - Quadratic Least Squares Moving Average

Description

Quadratic version of LSMA that fits parabolas instead of straight lines. Captures accelerating/decelerating trends better than linear regression. Less lag when the trend is accelerating. Output is the value of the quadratic polynomial fitted at the last bar. More responsive to nonlinear trends, used when the linear assumption is too limiting.

Syntax

var QLSMA(vars Series);
// or with a specific periodvar QLSMA(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the QLSMA indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate QLSMAvars Price = series(priceClose());var value = QLSMA(Price);var prevvalue = QLSMA(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "QLSMA", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.9.25. RET - Rate of Change Between Two Points

Description

Calculates percentage return between two specified points in time. More flexible than standard ROC because it can specify any pair of points instead of a fixed lookback. Useful for analyzing returns over custom horizons or for backtesting holding period optimization. The output is percentage change.

Syntax

var RET(vars Series);
// or with specific periodvar RET(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the RET indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate RETvars Price = series(priceClose());var value = RET(Price);var prevvalue = RET(Price)[1];
// Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "RET", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced RET parameters
- Always test on historical data before using in production

1.9.26. rising - Curve Rising

Description

Boolean function that returns true if the series is rising (current value > previous value). Applicable to prices, indicators, and volumes. Foundation for rule-based systems: "If price rises AND volume rises, then buy." Simple but powerful building block that, combined with other conditions, creates sophisticated entry logic.

Syntax

var rising(vars Series);
// or with specific periodvar rising(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the rising indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate rising vars Price = series(priceClose());var value = rising(Price);var prevvalue = rising(Price)[1]; // Previous value

// Log of value if(is(LOGFILE))printf("\n%s: %.4f", "rising", value);

// Example trading condition
if(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Incorporates volume information for more comprehensive analysis

1.9.27. risingF - Curve Rising Fuzzy

Description

Fuzzy version that returns the degree of rising (0-1). Considers the slope and consistency of the rise: steep consistent rise = 1, gradual rise = intermediate value. Lower slope or intermittent rise scores lower. Allows for quantified assessment instead of binary, useful for graduated responses and variable position sizing.

Syntax

var risingF(vars Series);
// or with specific periodvar risingF(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the risingF indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate risingF
vars Price = series(priceClose());
var value = risingF(Price);var prevvalue = risingF(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "risingF", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced risingF parameters.
- Always test on historical data before using in production

1.9.28. slope - Line Through Minima or Maxima

Description

Calculate slopes of regression lines through specific points (minimum, maximum, or all). It allows you to analyze swing lows vs. swing highs separately. The slope of lows ascending in an uptrend confirms support strength. Slope discrepancies between highs and lows can indicate patterns such as wedges and triangles.

Syntax

var slope(vars Series);
// or with specific periodvar slope(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the slope indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate slopevars Price = series(priceClose());var value = slope(Price);var prevvalue = slope(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "slope", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.9.29. SMAP - Average of Positive Values

Description

Average calculated only on positive values in a period, ignoring negative values and zeros. Useful for analyzing only upside performance or gaining days. Asymmetric analysis tool that can reveal different dynamics of positive vs. negative movements. Used in specialized indicators that treat gains and losses separately.

Syntax

var SMAP(vars Series);
// or with specific periodvar SMAP(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the SMAP indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate SMAPvars Price = series(priceClose());var value = SMAP(Price);var prevvalue = SMAP(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "SMAP", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced SMAP parameters
- Always test on historical data before using in production

1.9.30. Sum - Sum of Elements

Description

Calculates the sum of all values in a series over N periods. Different from the moving average (which divides by N). Useful for accumulation measures, total volume calculations, and when an absolute total is needed rather than an average. Foundation for many cumulative indicators and for calculations requiring totals.

Syntax

var Sum(vars Series);
// or with specific periodvar Sum(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the Sum indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate Sumvars Price = series(priceClose());var value = Sum(Price);var prevvalue =
Sum(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "Sum", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Incorporates volume information for more comprehensive analysis

1.9.31. SumDn - Sum of Falling Elements

Description

Sums only the negative movements (falling values) in the last N periods. Measures cumulative downside. Complementary to SumUp. The SumUp/SumDn ratio is analogous to RS in RSI calculation. Used for asymmetric analysis, separating upside from downside dynamics. Can reveal divergences between upside and downside momentum.

Syntax

```
var SumDn(vars Series);
// or with specific periodvar SumDn(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the SumDn indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate SumDnvars Price = series(priceClose());var value = SumDn(Price);var prevvalue = SumDn(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "SumDn", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {
// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Divergences with price can signal impending reversals

1.9.32. SumUp - Sum of Rising Elements

Description

Sum only of positive movements (rising values) in the last N periods. Measurements cumulative upside. Divergences between SumUp and price indicate weakening momentum. When SumUp declining but price flat or rising indicates fewer large gains compensating for more small gains. Asymmetric momentum measure.

Syntax

var SumUp(vars Series);
// or with specific periodvar SumUp(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the SumUp indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate SumUpvars Price = series(priceClose());var value = SumUp(Price);var prevvalue = SumUp(Price)[1]; // Previous value

```
// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "SumUp", value);

// Example trading condition
if(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Divergences with price can signal impending reversals

1.9.33. touch - Curve Touches Another

Description

Identifies when a series touches (comes extremely close to) another series without necessarily crossing. True when the distance is below the specified threshold. Useful for identifying tests of support/resistance levels. Different from crossovers because it can identify "near misses" that are still significant price action.

Syntax

```
var touch(vars Series);
// or with specific periodvar touch(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the touch indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate touch
vars Price = series(priceClose());
var value = touch(Price);var prevvalue = touch(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "touch", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}
```

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced touch parameters
- Always test on historical data before using in production

1.10. CURRENCY ANALYSIS AND CORRELATIONS

1.10.1. ccyMax - Strongest Forex Pair

Description

Identify the forex pair with the strongest momentum within a defined basket. Calculate the relative strength of each currency by compounding the movements of the major pairs. Useful for relative strength strategies that involve going long on the strongest pair. Can be used to build currency portfolios that exploit performance divergences between correlated currencies.

Syntax

var ccyMax(vars Series);
// or with specific periodvar ccyMax(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ccyMax indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ccyMaxvars Price = series(priceClose());var value = ccyMax(Price);var prevvalue = ccyMax(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ccyMax", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Divergences with price can signal impending reversals

1.10.2. ccyMin - Weakest Forex Pair

Description

Identifies the forex pair with the weakest momentum within a defined basket. Opposite of ccyMax, useful for identifying short opportunities or constructing long-short currency spreads. Combined with ccyMax, it allows you to implement forex pair trading strategies by exploiting temporary divergences between currencies that normally move together.

Syntax

var ccyMin(vars Series);
// or with specific periodvar ccyMin(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ccyMin indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ccyMinvars Price = series(priceClose());var value = ccyMin(Price);var prevvalue = ccyMin(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ccyMin", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Divergences with price can signal impending reversals

1.10.3. ccyReset - Initialize Currency Strength

Description

Resets the currency strength calculation system to its initial conditions. This function must be called at the start of new analyses or when changing strength calculation parameters. It clears internal arrays and reinitializes baselines for comparison, ensuring that subsequent strength values are calculated consistently and not influenced by previous data.

Syntax

var ccyReset(vars Series);
// or with specific periodvar ccyReset(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ccyReset indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ccyResetvars Price = series(priceClose());var value = ccyReset(Price);var prevvalue = ccyReset(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ccyReset", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();
}
}

Usage Notes

- See the Zorro documentation for advanced ccyReset parameters.
- Always test on historical data before using in production

1.10.4. ccySet - Define Currency Strength

Description

Allows you to manually set or update strength values for specific currencies. Useful for incorporating fundamental or sentiment analysis into quantitative strategies, or for overriding automatic calculations with proprietary valuations. It allows you to implement hybrid models that combine technical indicators with fundamental or alternative factors.

Syntax

var ccySet(vars Series);
// or with specific periodvar ccySet(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ccySet indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ccySetvars Price = series(priceClose());var value = ccySet(Price);var prevvalue = ccySet(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "ccySet", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible long signalif
(!NumOpenLong)
enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible short signalif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced ccySet parameters
- Always test on historical data before using in production

1.10.5. ccyStrength - Get Currency Strength

Description

Retrieves the current relative strength value for a specific currency. It returns a normalized value representing the currency's overall momentum relative to a basket. Positive indicates relative

strength, negative indicates weakness. Essential for building trading signals based on currency strength divergences and for implementing currency rotation strategies.

Syntax

var ccyStrength(vars Series);
// or with specific periodvar ccyStrength(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the ccyStrength indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate ccyStrengthvars Price = series(priceClose());var value = ccyStrength(Price);var prevvalue = ccyStrength(Price)[1]; // Previous value

// Log of value
if(is(LOGFILE))printf("\n%s: %.4f", "ccyStrength", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Divergences with price can signal impending reversals

1.11. 📈 COT (COMMITMENT OF TRADERS)

1.11.1. COT - Commitment Of Traders Report

Description

Access data from the COT (Commitment of Traders) report published weekly by the CFTC, showing aggregate positions of commercials, large speculators, and small traders in futures. Useful for analyzing institutional sentiment and identifying positioning extremes. Commercials tend to be right at turning points, so extreme positions can anticipate trend reversals.

Syntax

var COT(vars Series);
// or with specific periodvar COT(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the COT indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate COTvars Price = series(priceClose());var value = COT(Price);var prevvalue = COT(Price)[1];
// Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "COT", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends

1.11.2. COTCommercialPos - COT Commercials Net Position

Description

Extracts the net position (long - short) of commercial traders from the COT report. Commercials are hedgers (producers, traders) considered "smart money." Extremely long net positions can indicate market bottoms, while extremely short positions indicate possible tops. Used as a contrarian indicator: when commercials are at their highest long positions and speculators are at their highest short positions, it's often time to buy.

Syntax

var COTCommercialPos(vars Series);
// or with specific periodvar COTCommercialPos(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the COTCommercialPos indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate COTCommercialPosvars Price = series(priceClose());var value = COTCommercialPos(Price);var prevvalue = COTCommercialPos(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "COTCommercialPos", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- See the Zorro documentation for advanced COTCommercialPos parameters.
- Always test on historical data before using in production

1.11.3. COTCommercialIndex - COT Index

Description

Normalizes COT commercial positions into a 0-100 percentile index based on historical range. Above 80 indicates extremely bullish commercial positioning (historically near the bottom), while below 20 indicates extremely bearish (near the top). Easier to interpret than absolute positions, it allows comparisons across different commodities and timeframes.

Syntax

```
var COTCommercialIndex(vars Series);
// or with specific periodvar COTCommercialIndex(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the COTCommercialIndex indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate COTCommercialIndexvars Price = series(priceClose());var value = COTCommercialIndex(Price);var prevvalue = COTCommercialIndex(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "COTCommercialIndex", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced parameters.

COTCommercialIndex

- Always test on historical data before using in production

1.11.4. COTOpenInterest - COT Open Interest

Description

Access total open interest data from COT reports. Rising open interest during a trend confirms trend strength (new money flow); decreasing open interest may indicate profit-taking and a possible reversal. Divergences between price and open interest are significant: price up but OI down suggests a weak trend due to short-covering, not new purchases.

Syntax

var COTOpenInterest(vars Series);
// or with specific periodvar COTOpenInterest(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the COTOpenInterest indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate COTOpenInterestvars Price = series(priceClose());var value = COTOpenInterest(Price);var prevvalue = COTOpenInterest(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "COTOpenInterest", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)
enterLong();
}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends
- Divergences with price can signal impending reversals

1.12. 🎯 SENTIMENT AND MARKET STATE

1.12.1. CBI - Cold Blood Index

Description

Proprietary indicator that measures the market's "coolness" or emotionality based on the dispersion of returns and volume. High values indicate a rational and calm market, while low values indicate an emotional and volatile market. Useful for identifying optimal market conditions for specific strategies: mean-reversion strategies prefer a low CBI, trend-following strategies prefer a high CBI.

Syntax

var CBI(vars Series);
// or with a specific period var CBI(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the CBI indicator for the current period

Example of Use

```
function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate CBIvars Price = series(priceClose());
var value = CBI(Price);
var prevvalue = CBI(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "CBI", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal
longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal
shortif(!NumOpenShort)enterShort();}}
```

Usage Notes

- Particularly effective for identifying and confirming trends
- Incorporates volume information for more comprehensive analysis

1.12.2. IBS - Internal Bar Strength

Description

Measures where the close is positioned within the bar's range: (Close-Low)/(High-Low). Values close to 0 indicate a close near a low (weakness), while values close to 1 indicate a close near a high (strength). Used in mean reversion: a very low IBS on a down bar in an uptrend suggests an oversold condition and a possible rebound. Particularly effective on equity for short-term reversal plays.

Syntax

```
var IBS(vars Series);
// or with specific periodvar IBS(vars Series, int Period);
```

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the IBS indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate IBSvars Price = series(priceClose());var value = IBS(Price);var prevvalue = IBS(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "IBS", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- Particularly effective for identifying and confirming trends
- Useful for identifying overbought/oversold conditions

1.12.3. SentimentLW - Williams' Market Sentiment

Description

Larry Williams's proprietary sentiment indicator based on market internals and patterns. Measures the market's emotional temperature: extreme readings indicate panic (buy opportunity) or euphoria (sell opportunity). Contrarian indicator: when everything is bullish, it's time to sell; everything is bearish, it's time to buy. Useful gauge of market psychology extremes.

Syntax

var SentimentLW(vars Series);
// or with specific periodvar SentimentLW(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the SentimentLW indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate SentimentLWvars Price = series(priceClose());var value = SentimentLW(Price);var prevvalue = SentimentLW(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "SentimentLW", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced SentimentLW parameters.
- Always test on historical data before using in production

1.12.4. SentimentG - Genesis Sentiment Index

Description

Proprietary sentiment indicator that aggregates multiple sentiment sources: put/call ratios, VIX, advance/decline, etc. Seek to identify market extremes based on behavioral finance principles. High readings indicate excessive optimism (bearish), low excessive pessimism (bullish). Comprehensive sentiment gauge for timing reversals at market extremes.

Syntax

var SentimentG(vars Series);
// or with specific periodvar SentimentG(vars Series, int Period);

Parameters

Series (vars)

Input time series on which to calculate the indicator

Example: priceClose(), priceHigh(), custom series

Period (int) - Optional

Number of periods for calculating the indicator

Typical values: 14 (default), 20, 50, 200

Return Value

var - Calculated value of the SentimentG indicator for the current period

Example of Use

function run() {
BarPeriod = 60; // Timeframe 1 hour

// Calculate SentimentGvars Price = series(priceClose());var value = SentimentG(Price);var prevvalue = SentimentG(Price)[1]; // Previous value

// Log of valueif(is(LOGFILE))printf("\n%s: %.4f", "SentimentG", value);

// Example trading conditionif(value > 0 && prevvalue <= 0) {// Positive crossover - possible signal longif(!NumOpenLong)enterLong();}

if(value < 0 && prevvalue >= 0) {// Negative crossover - possible signal shortif(!NumOpenShort)enterShort();}}

Usage Notes

- See the Zorro documentation for advanced SentimentG parameters.
- Always test on historical data before using in production

2. CHAPTER 2: MARKETS & TRADING

2.1. ASSET MANAGEMENT

2.1.1. asset - Asset Selection

Description:

Select the current asset for trading. All subsequent commands (prices, orders, indicators) refer to this asset.

Syntax:

bool asset(string Name);

bool asset(string Name);

Parameters:\

- Name (string): Name of the asset to select (e.g. "EUR/USD", "SPY", "GC")

  Return Value: bool - TRUE if asset found and selected correctly, FALSE otherwise

  Example:

Parameters:

- **Name** (string): Name of the asset to select (e.g. "EUR/USD", "SPY", "GC")

Return Value:

bool - TRUE if asset found and selected correctly, FALSE otherwise

Example:

function run() {
// Select single assetif(!asset("EUR/USD")) {printf("\nERROR: Asset EUR/USD not found");return;}

// Now all functions refer to EUR/USDvar price = priceClose();enterLong();}

// Multi-asset trading
function run() {
// Loop over multiple assetswhile(asset(loop("EUR/USD", "GBP/USD", "USD/JPY"))) {vars Price = series(priceClose());var sma = SMA(Price, 50);

if(Price[0] > sma)enterLong();else if(Price[0] < sma)enterShort();}}

```
function run() {
// Select single asset
if(!asset("EUR/USD")) {
printf("\nERROR: Asset EUR/USD not found");
return;
}
// Now all functions refer to EUR/USD
var price = priceClose();
enterLong();
}
// Multi-asset trading
function run() {
// Loop across multiple assets
while(asset(loop("EUR/USD", "GBP/USD", "USD/JPY"))) {
vars Price = series(priceClose());
var sma = SMA(Price, 50);
if(Price[0] > sma)
enterLong();
else if(Price[0] < sma)
enterShort();
}
}
```

Note:\

- Asset must be present in the .csv file or broker\
- The selected asset remains active until the next asset() call.
- Using with loop() for multi-asset trading\
- Always verify the return value in LIVE mode

---

assetAdd - Add Asset to List

Description: Adds an asset to the list of available assets with all its parameters. Useful for dynamically creating assets or modifying existing assets without editing the CSV file.

Syntax:

Note:

- Asset must be present in the .csv file or broker
- The selected asset remains active until the next asset() call.

- Using with loop() for multi-asset trading
- Always check the return value in LIVE mode

2.1.2. assetAdd - Add Asset to List

Description:

Adds an asset to the list of available assets with all its parameters. Useful for dynamically creating assets or modifying existing assets without editing the CSV file.

Syntax:

int assetAdd(string Name, var Price, var Spread, var RollLong, var RollShort,
var Pip, var PipCost, var MarginCost, var Leverage,var LotAmount, var Commission, string Symbol);

int assetAdd(string Name, var Price, var Spread, var RollLong, var RollShort,

var Pip, var PipCost, var MarginCost, var Leverage,

var LotAmount, var Commission, string Symbol);

Parameters:\

- Name (string): Name of the asset\
- Price (var): Initial price (0 = keep existing)\
- Spread (var): Spread in price units\
- RollLong (var): Swap long overnight\
- RollShort (var): Swap short overnight\
- Pip (var): Size of the pip\
- PipCost (var): Cost per pip
- MarginCost (var): Margin per unit\
- Leverage (var): Financial leverage\
- LotAmount (var): Units per lot\
- Commission (var): Commission per lot\
- Symbol (string): Symbol for the broker (format "Source:Symbol")

  Return Value:int - Number of assets in the list after adding

  Example:

Parameters:

- **Name** (string): Name of the asset
- **Price** (var): Initial price (0 = keep existing)
- **Spread** (var): Spread in price units
- **RollLong** (var): Long overnight swap
- **RollShort** (var): Short overnight swap
- **Pip** (var): Size of the pip
- **PipCost** (var): Cost of the pip

- **MarginCost** (var): Margin per unit
- **Leverage** (var): Financial leverage
- **LotAmount** (var): Units per lot
- **Commission** (var): Commission per lot
- **Symbol** (string): Symbol for the broker (format "Source:Symbol")

Return Value:

int - Number of assets in the list after adding

Example:

```
function main() {
// Add SPY with custom parameters assetAdd("SPY", 450, 0.01, 0, 0, 0.01, 1, 450, 4, 1, 0.5, "STOOQ:SPY.US");

// Add synthetic asset assetAdd("USDBASKET", 100, 0, 0, 0, 0.01, 1, 0, 1, 1, 0, "");

// Change only the symbol of an existing asset if(is(INITRUN)) { assetList("MyAssetList.csv"); for(listedassets) assetAdd(Asset, 0, 0, 0, 0, 0, 0, 0, 0, 0, strf("NewBroker:%s", SymbolLive)); }}
function main() {
// Add SPY with custom parameters
assetAdd("SPY", 450, 0.01, 0, 0, 0.01, 1, 450, 4, 1, 0.5, "STOOQ:SPY.US");
// Add synthetic asset
assetAdd("USDBASKET", 100, 0, 0, 0, 0.01, 1, 0, 1, 1, 0, "");
// Only change the symbol of an existing asset
if(is(INITRUN)) {
assetList("MyAssetList.csv");
for(listedassets)
assetAdd(Asset, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
strf("NewBroker:%s", SymbolLive));
}
}
```

Note:\

- Parameters set to 0 retain existing values.
- The symbol may include the data source (e.g., "STOOQ:SPY.US")\
- Useful for creating synthetic assets or baskets\
- It can be used to edit list assets\
- Must be called in INITRUN for persistent assets

---

assetHistory - Download Historical Prices

Description: Downloads historical prices from brokers or online sources (Stooq, Yahoo, AlphaVantage, Quandl, etc.) and saves them to the History folder.

Syntax:

Note:

- Parameters set to 0 retain existing values
- The symbol may include the data source (e.g., "STOOQ:SPY.US")
- Useful for creating synthetic assets or baskets
- It can be used to edit assets in the list
- Must be called in INITRUN for persistent assets

2.1.3. assetHistory - Download Price History

Description:

Download historical prices from brokers or online sources (Stooq, Yahoo, AlphaVantage, Quandl, etc.) and save them in the History folder.

Syntax:

int assetHistory(string Name, int Mode);

int assetHistory(string Name, int Mode);

Parameters:\

- Name (string): Name of the asset to download\
- Mode (int): Download mode\
- 0 = from broker (M1/tick data)\
- 1 = from broker (EOD data)\
- FROMSTOOQ = from Stooq (EOD)\
- FROMYAHOO = from Yahoo Finance (EOD)\
- FROMAV = from AlphaVantage (requires API key)\
- FROMQUANDL = from Quandl (requires API key)\
- FROMSOURCE = from source configured with assetSource()\
- UNADJUSTED = not adjusted for splits/dividends (can be combined with |)

Return Value: int - Number of bars downloaded, 0 if error

Example:

Parameters:

- **Name** (string): Name of the asset to download
- **Mode** (int): Download mode
  - 0 = from broker (M1/tick data)
  - 1 = from broker (EOD data)

- FROMSTOOQ = from Stooq (EOD)
- FROMYAHOO = from Yahoo Finance (EOD)
- FROMAV = from AlphaVantage (requires API key)
- FROMQUANDL = from Quandl (requires API key)
- FROMSOURCE = from source configured with assetSource()
- UNADJUSTED = not adjusted for splits/dividends (can be combined with |)

Return Value:

int - Number of bars downloaded, 0 if error

Example:

```
// Update M1 price history of all assets from the broker
function main() {NumYears = 2;while(loop(Assets))assetHistory(Loop1, 1);}

// Download D1 prices from Quandlstring Name;while(Name = loop("AAPL", "MSFT", "GOOGL"))
assetHistory(strf("WIKI/%s", Name), FROMQUANDL);

// Download and plot Bitcoin prices from Bitfinexfunction run() {BarPeriod =
1440;assetHistory("BITFINEX/BTCUSD",
FROMQUANDL);assetAdd("BTCUSD");asset("BTCUSD");set(PLOTNOW);}

// Download unadjusted dataHistory = "*una.t6";assetHistory("SPY", FROMSTOOQ|UNADJUSTED);

// Using assetSource for custom
sourceassetSource("AAPL.t6","https://stooq.pl/q/d/l/?s=AAPL.US&d1=20000101&d2=20201024&i=
d",0, 0, "+%Y-%m-%d,f3,f1,f2,f4,f6");assetHistory(0, FROMSOURCE);
// Update M1 price history of all assets from the broker
function main() {
NumYears = 2;
while(loop(Assets))
assetHistory(Loop1, 1);
}
// Download D1 prices from Quandl
string Name;
while(Name = loop("AAPL", "MSFT", "GOOGL"))
assetHistory(strf("WIKI/%s", Name), FROMQUANDL);
// Download and plot Bitcoin prices from Bitfinex
function run() {
BarPeriod = 1440;
assetHistory("BITFINEX/BTCUSD", FROMQUANDL);
assetAdd("BTCUSD");
asset("BTCUSD");
```

```
set(PLOTNOW);
}
// Download unadjusted data
History = "*una.t6";
assetHistory("SPY", FROMSTOOQ|UNADJUSTED);
// Using assetSource for custom source
assetSource("AAPL.t6",
"https://stooq.pl/q/d/l/?s=AAPL.US&d1=20000101&d2=20201024&i=d",
0, 0, "+%Y-%m-%d,f3,f1,f2,f4,f6");
assetHistory(0, FROMSOURCE);
```

Note:\

- Prices are saved in History/ with extension .t6 (EOD) or .t1 (tick/M1)\
- M1 data is split by year, EOD in a single file\
- Use UpdateDays for automatic download at the start of the backtest\
- Existing data is extended, not overwritten.
- Check History/history.csv or history.json for download errors.
- The Quandl database name and Stooq's ".US" are automatically removed.
- Characters '/', '.', and ':' are removed from filenames.

---

assetList - Load Asset List

Description: Loads a list of assets from a CSV file. The list defines which assets are available for trading and their properties.

Syntax:

Note:

- Prices are saved in History/ with extension .t6 (EOD) or .t1 (tick/M1)
- M1 data is split by year, EOD in a single file
- Use UpdateDays for automatic download at the start of your backtest
- Existing data is extended, not overwritten
- Check History/history.csv or history.json for download errors
- The Quandl database name and Stooq ".US" are automatically removed
- Characters '/', '.', ':' are removed from file names

2.1.4. assetList - Load Asset List

Description:

Upload a list of assets from a CSV file. The list defines which assets are available for trading and their properties.

Syntax:

bool assetList(string Filename);
int assetList(string Filename, int Select); // Extended version

bool assetList(string Filename);

int assetList(string Filename, int Select); // Extended version

Parameters:\

- Filename (string): Name of the CSV file (without extension) in the History/\ folder
- Select (int): [Optional] Asset to select (0 = default, -1 = none)

  Return Value: bool/int - TRUE/asset number if loaded correctly, FALSE/0 otherwise

  CSV File Format:

Parameters:

- **Filename** (string): Name of the CSV file (without extension) in the History/ folder

- **Select** (int): [Optional] Asset to select (0 = default, -1 = none)

Return Value:

bool/int - TRUE/asset number if loaded successfully, FALSE/0 otherwise

CSV File Format:

Name,Price,Spread,RollLong,RollShort,PIP,PIPCost,MarginCost,Leverage,LotAmount,Commission,Symbol
EUR/USD,1,0.00001,-0.11,0.09,0.0001,1,1000,100,10000,0,""GBP/USD,1,0.00002,-0.09,0.07,0.0001,1,1000,100,10000,0,""SPY,450,0.01,0,0,0.01,1,450,4,1,0.5,"STOOQ:SPY.US"

Name,Price,Spread,RollLong,RollShort,PIP,PIPCost,MarginCost,Leverage,LotAmount,Commission,Symbol

EUR/USD,1,0.00001,-0.11,0.09,0.0001,1,1000,100,10000,0,""

GBP/USD,1,0.00002,-0.09,0.07,0.0001,1,1000,100,10000,0,""

SPY,450,0.01,0,0,0.01,1,450,4,1,0.5,"STOOQ:SPY.US"

Example:

Example:

```
function run() {
// Load asset list for forexassetList("AssetsFix"); // Read History/AssetsFix.csv

// Load custom listassetList("MyAssets");

// Loop through all assets in the listfor(asset(loop(Assets))) {printf("\nTrading: %s", Asset);// ... trading logic ...}
}
```

```
// Load without selecting assets
assetList("Portfolio", -1);
function run() {
// Load Forex Asset List
assetList("AssetsFix"); // Read History/AssetsFix.csv
// Load custom list
assetList("MyAssets");
// Loop through all assets in the list
for(asset(loop(Assets))) {
printf("\nTrading: %s", Asset);
// ... trading logic ...
}
}
// Load without selecting assets
assetList("Portfolio", -1);
```

Note:\

- Required columns: Name, Price, Spread, PIP, PIPCost\
- For options use "AssetsIB" with IB broker\
- The file is loaded only once at the beginning\
- NumAssetsListed contains the number of loaded assets\
- Default: AssetsFix.csv if not specified
  ---# ADVANCED ASSET MANAGEMENTassetType - Asset Type IdentificationDescription:Determines the type of an asset based on its name and characteristics.Syntax:

Note:

- Required columns: Name, Price, Spread, PIP, PIPCost
- For options use "AssetsIB" with IB broker
- The file is loaded only once at the beginning
- NumAssetsListed contains the number of loaded assets
- Default: AssetsFix.csv if not specified

2.2. ADVANCED ASSET MANAGEMENT

2.2.1. assetType - Asset Type Identification

Description:

Determine the type of an asset based on its name and characteristics.

Syntax:

int assetType(string Name);

int assetType(string Name);

Parameters:\

- Name (string): Name of the asset to analyze

  Return Value:int - Asset type:\
- 1 = FOREX (currency pair)\
- 2 = CFD (Contract For Difference)\
- 3 = STOCK (action)\
- 4 = FUTURE (futures contract)\
- 0 = unknown/other

  Example:

Parameters:

- **Name** (string): Name of the asset to analyze

Return Value:

int - Asset Type:

- 1 = FOREX (currency pair)
- 2 = CFD (Contract For Difference)
- 3 = STOCK (action)
- 4 = FUTURE (futures contract)
- 0 = unknown/other

Example:

```
function run() {
asset("EUR/USD");if(assetType(Asset) == 1) // FOREXprintf("\n%s is a Forex pair", Asset);

asset("SPY");if(assetType(Asset) == 3) // STOCKprintf("\n%s is a stock", Asset);

// Loop with type filter for(asset(loop(Assets))) {if(assetType(Asset) == 1) { // Forex only// Trading logic for forex...}}}
function run() {
asset("EUR/USD");
if(assetType(Asset) == 1) // FOREX
printf("\n%s is a Forex pair", Asset);
asset("SPY");
if(assetType(Asset) == 3) // STOCK
printf("\n%s is a stock", Asset);
// Loop with filter type
for(asset(loop(Assets))) {
if(assetType(Asset) == 1) { // Forex only
// Trading logic for forex...
```

}

}

}

Note:\

- Recognizes common patterns in names (e.g. "/" for forex)\
- May be inaccurate for non-standard names\
- Useful for applying different logics for each asset type.
- Does not require asset selection

---

assetSelect - Optimal Asset Selection

Description: Automatically selects the best N assets from the list based on performance, volatility, or momentum criteria.

Syntax:

Note:

- Recognizes common patterns in names (e.g. "/" for forex)
- May be inaccurate for non-standard names
- Useful for applying different logics for each asset type
- Does not require the asset to be selected

2.2.2. assetSelect - Optimal Asset Selection

Description:

Automatically selects the best N assets from the list based on performance, volatility, or momentum criteria.

Syntax:

int assetSelect(int Num, int Mode);

int assetSelect(int Num, int Mode);

Parameters:\

- Num (int): Number of assets to select\
- Mode (int): Selection Criteria\
- 0 = Performance (ROC)\
- 1 = Volatility (ATR/StdDev)\
- 2 = Momentum (combined)\
- 4 = Inverse (worst performers)

Return Value:int - Number of selected assets

Example:

Parameters:

- **Num** (int): Number of assets to select
- **Mode** (int): Selection criterion
  - 0 = Performance (ROC)
  - 1 = Volatility (ATR/StdDev)
  - 2 = Momentum (combined)
  - 4 = Inverse (worst performers)

Return Value:

int - Number of selected assets

Example:

```
// ETF Rotation Strategy
function run() {BarPeriod = 1440; // DailyLookBack = 252; // 1 year

if(tdm() == 1) { // First day of the month // Select the 3 best performing assets assetSelect(3, 0);

// Close positions on unselected assets for(opentrades) { if(!is(SELECTED))exitTrade();}

// Open positions on selected assets for(selectedassets) {
asset(LoopAsset);
enterLong();}}}

// Rotation with inverse selection (worst performers) assetSelect(2, 4); // Select the 2 worst performers for short for(selectedassets) { asset(LoopAsset); enterShort();}
// ETF Rotation Strategy
function run() {
BarPeriod = 1440; // Daily
LookBack = 252; // 1 year
if(tdm() == 1) { // First day of the month
// Select the 3 best performing assets
assetSelect(3, 0);
// Close positions on unselected assets
for(opentrades) {
if(!is(SELECTED))
exitTrade();
}
// Open positions on selected assets
for(selectedassets) {
asset(LoopAsset);
```

```
enterLong();
}
}
}
// Rotation with inverse selection (worst performers)
assetSelect(2, 4); // Select the 2 worst for short
for(selectedassets) {
asset(LoopAsset);
enterShort();
}
```

Note:\

- Requires LookBack > 0 to calculate performance\
- Set the SELECTED flag on the selected assets\
- Use selectedassets in loop to iterate\
- Combinable with algo() for multi-strategy\
- Performance calculated over LookBack period

---

assetSource - Custom Data Source Configuration

Description: Configure parameters to download data from a custom online source (REST API, website) for use with assetHistory().

Syntax:

Note:

- Requires LookBack > 0 to calculate performance
- Set the SELECTED flag on the selected assets
- Use selectedassets in the loop to iterate
- Combinable with algo() for multi-strategy
- Performance calculated on LookBack period

2.2.3. assetSource - Custom Data Source Configuration

Description:

Configure parameters to download data from a custom online source (REST API, website) for use with assetHistory().

Syntax:

```
void assetSource(string Name, string URL, int Start, int End, string Format);
void assetSource(string Name, string URL, int Start, int End, string Format);
```

Parameters:\

- Name (string): Output file name (e.g. "AAPL.t6")\
- URL (string): URL of the data resource\
- Start (int): Start date YYYYMMDD (0 = from StartDate)\
- End (int): End Date YYYYMMDD (0 = to EndDate)\
- Format (string): Data parsing format\
- "+%Y-%m-%d,f3,f1,f2,f4,f6" for CSV with date,O,H,L,C,V\
- "*JSON:field.subfield" for JSON\
- Prefixes: '+' = skip header, '*' = special format

  Return Value: void

  Example:

Parameters:

- **Name** (string): Output file name (e.g. "AAPL.t6")
- **URL** (string): URL of the data resource
- **Start** (int): Start date YYYYMMDD (0 = from StartDate)
- **End** (int): End date YYYYMMDD (0 = to EndDate)
- **Format** (string): Data parsing format
  - "+%Y-%m-%d,f3,f1,f2,f4,f6" for CSV with date,O,H,L,C,V
  - "*JSON:field.subfield" for JSON
  - Prefixes: '+' = skip header, '*' = special format

Return Value:

void

Example:

```
// Download from Stooq with custom format
assetSource("AAPL.t6","https://stooq.pl/q/d/l/?s=AAPL.US&d1=20000101&d2=20201024&i=d",0,
0,"+%Y-%m-%d,f3,f1,f2,f4,f6"); // Skip header, then Date,O,H,L,C,VassetHistory(0, FROMSOURCE);

// Download Bitcoin OHLC from Coin.io JSONint Month, Year = 2019, MinutesPerMonth =
31*1440;for(Month = 1; Month <= 12; Month++) {string URL =
strf("https://rest.coinapi.io/v1/ohlcv/BTC/USD/history?periodid=1MIN&timestart=%d-%02d-
01&limit=%d",Year, Month, MinutesPerMonth);

assetSource("BTC.t1",URL,ymd(Year, Month, 1),ymd(Year, Month,
31),"*JSON:timeperiodstart,priceopen,pricehigh,pricelow,priceclose,volumetraded");

assetHistory(0, FROMSOURCE);}

// custom CSV with ; as
```

```
separatorassetSource("Custom.t6","http://myserver.com/data.csv",20200101,
20201231,"+;%Y%m%d;f1;f2;f3;f4;f5"); // Date;O;H;L;C;V
// Download from Stooq with custom format
assetSource("AAPL.t6",
"https://stooq.pl/q/d/l/?s=AAPL.US&d1=20000101&d2=20201024&i=d",
0, 0,
"+%Y-%m-%d,f3,f1,f2,f4,f6"); // Skip header, then Date,O,H,L,C,V
assetHistory(0, FROMSOURCE);
// Download Bitcoin OHLC from Coin.io JSON
int Month, Year = 2019, MinutesPerMonth = 31*1440;
for(Month = 1; Month <= 12; Month++) {
string URL = strf(
"https://rest.coinapi.io/v1/ohlcv/BTC/USD/history?periodid=1MIN&timestart=%d-%02d-
01&limit=%d",
Year, Month, MinutesPerMonth);
assetSource("BTC.t1",
URL,
ymd(Year, Month, 1),
ymd(Year, Month, 31),
"*JSON:timeperiodstart,priceopen,pricehigh,pricelow,priceclose,volumetraded");
assetHistory(0, FROMSOURCE);
}
// Custom CSV with ; as separator
assetSource("Custom.t6",
"http://myserver.com/data.csv",
20200101, 20201231,
"+;%Y%m%d;f1;f2;f3;f4;f5"); // Date;O;H;L;C;V
```

Note:\

- Use with assetHistory(0, FROMSOURCE)\
- Format supports wildcards: %Y=year, %m=month, %d=day, %H=hour, %M=minute\
- 'f1' = first numeric field, 'f2' = second, etc.\
- JSON: Use dot notation for nested fields\
- URL may contain authentication parameters\
- Data saved in History/ with specified name\
- Supports automatic HTTP redirects

PRICE FUNCTIONS

price - Average Bar Price

Description:
Returns the average price (open+high+low+close)/4 of the current or specified bar.

Syntax:

Note:

- Use with assetHistory(0, FROMSOURCE)
- Format supports wildcards: %Y=year, %m=month, %d=day, %H=hour, %M=minute
- 'f1' = first numeric field, 'f2' = second, etc.
- JSON: Use dot notation for nested fields
- URL may contain authentication parameters
- Data saved in History/ with specified name
- Supports automatic HTTP redirects

2.3. PRICE FUNCTIONS

2.3.1. price - Average Bar Price

Description:

Returns the average price (open+high+low+close)/4 of the current or specified bar.

Syntax:

var price(int Offset);

var price(int Offset);

Parameters:\

- Offset (int): Number of bars back (0 = current, 1 = previous, etc.)

Return Value:var - Average price at the specified bar

Example:

Parameters:

- **Offset** (int): Number of bars back (0 = current, 1 = previous, etc.)

Return Value:

var - Average price at the specified bar

Example:

```
function run() {
var currentPrice = price(0); // Current price var yesterdayPrice = price(1); // Yesterday

// Price comparison if(price(0) > price(1))
```

```
printf("\nPrice is rising");
}
function run() {
var currentPrice = price(0); // Current price
var yesterdayPrice = price(1); // Yesterday
// Price comparison
if(price(0) > price(1))
printf("\nPrice is rising");
}
```

---

priceOpen, priceClose, priceHigh, priceLow

Description: Returns the OHLC (Open, High, Low, Close) prices of the current or specified bar.

Syntax:

- **priceOpen, priceClose, priceHigh, priceLow**

Description:

Returns the OHLC (Open, High, Low, Close) prices of the current or specified bar.

Syntax:

```
var priceOpen(int Offset);
var priceHigh(int Offset);var priceLow(int Offset);var priceClose(int Offset);
var priceOpen(int Offset);
var priceHigh(int Offset);
var priceLow(int Offset);
var priceClose(int Offset);
```

Parameters:\

- Offset (int): Number of bars back (0 = current, 1 = previous, etc.)

  Return Value:var - Asking price at the specified bar

  Example:

Parameters:

- **Offset** (int): Number of bars back (0 = current, 1 = previous, etc.)

Return Value:

var - Asking price at the specified bar

Example:

```
function run() {
// Current bar pricesvar o = priceOpen(0);var h = priceHigh(0);var l = priceLow(0);var c =
```

```
priceClose(0);

// Candlestick pattern detectionvar body = c - o;var range = h - l;

if(abs(body) < 0.3 * range)printf("\nDoji candle detected");

// True Rangevar tr = max(h - l,abs(h - priceClose(1)),abs(l - priceClose(1)));}
function run() {
// Current bar prices
var o = priceOpen(0);
var h = priceHigh(0);
var l = priceLow(0);
var c = priceClose(0);
// Candlestick pattern detection
var body = c - o;
var range = h - l;
if(abs(body) < 0.3 * range)
printf("\nDoji candle detected");
// True Range
var tr = max(h - l,
abs(h - priceClose(1)),
abs(l - priceClose(1)));
}
```

Note:\

- In backtest: data from History file\
- Live: Latest data from brokers\
- Negative offset = future (only in test mode with peek)

ADVANCED PRICE FUNCTIONS

priceQuote - Real-Time Price

Description: In LIVE mode, returns the last tick price received from the broker. In backtest, returns priceClose(0). Used to simulate HFT and tick-based strategies.

Syntax:

Note:

- In backtest: data from History file
- Live: Latest data from brokers

- Negative offset = future (only in test mode with peek)

- **ADVANCED PRICE FUNCTIONS**

2.3.2. priceQuote - Real-Time Price

Description:

In LIVE mode, it returns the last tick price received from the broker. In backtest mode, it returns priceClose(0). Used to simulate HFT and tick-based strategies.

Syntax:

var priceQuote();
var priceQuote(int Type); // Version with type

var priceQuote();

var priceQuote(int Type); // Version with type

Parameters:\

- Type (int): [Optional]\
- 0 or omitted = last price\
- 1 = bid\
- 2 = ask\
- 3 = mid = (bid+ask)/2

  Return Value:var - Last tick price received

  Example:

Parameters:

- **Type** (int): [Optional]

- 0 or omitted = last price
- 1 = bid
- 2 = ask
- 3 = mid = (bid + ask) / 2

Return Value:

var - Last tick price received

Example:

// HFT tick-based strategy
function tick() {var bid = priceQuote(1);var ask = priceQuote(2);var spread = ask - bid;

if(spread < 0.0002) // Spread tightenerLong();}

// Simulate tick in backtestfunction run() {// Simulate 10 ticks per barrafor(int i = 0; i < 10; i++) {// Generate random price between Low and Highvar tickPrice = priceLow(0) +(priceHigh(0) - priceLow(0)) * (i+1)/10.0;priceQuote(tickPrice); // Set tick price

```
// Trading logic based on tickif(tickPrice > SomeLevel)enterLong();}}
// HFT tick-based strategy
function tick() {
var bid = priceQuote(1);
var ask = priceQuote(2);
var spread = ask - bid;
if(spread < 0.0002) // Spread tight
enterLong();
}
// Tick simulation in backtest
function run() {
// Simulates 10 ticks per bar
for(int i = 0; i < 10; i++) {
// Generate random price between Low and High
var tickPrice = priceLow(0) +
(priceHigh(0) - priceLow(0)) * (i+1)/10.0;
priceQuote(tickPrice); // Set tick price
// Tick-based trading logic
if(tickPrice > SomeLevel)
enterLong();
}
}
```

Note:\

- In LIVE requires broker with tick data\
- In backtest it must be fed manually with priceSet()\
- Use tick() function for tick-based strategies\
- marketVal(0) returns the spread of the last tick

---

priceSet - Manual Price Edit

Description: Manually sets a price in the price history. Used to create synthetic assets, simulate scenarios, or correct data.

Syntax:

Note:

- In LIVE requires broker with tick data

- In backtest it must be fed manually with priceSet()
- Use tick() function for tick-based strategies
- marketVal(0) returns the spread of the last tick

2.3.3. priceSet - Manual Price Change

Description:

Manually set a price in the price history. Used to create synthetic assets, simulate scenarios, or correct data.

Syntax:

void priceSet(int Type, var Value, int Offset);

void priceSet(int Type, var Value, int Offset);

Parameters:\

- Type (int): Price type to set\
- 0 = Open\
- 1 = High\
- 2 = Low\
- 3 = Close\
- 4 = Tick (for priceQuote)\
- Value (var): New price value\
- Offset (int): Bar to be modified (0 = current, negative = future)

  Return Value: void

  Example:

Parameters:

- **Type** (int): Price type to set
  - 0 = Open
  - 1 = High
  - 2 = Low
  - 3 = Close
  - 4 = Tick (for priceQuote)
- **Value** (var): New price value
- **Offset** (int): Bar to be modified (0 = current, negative = future)

Return Value:

void

Example:

```
// Create flat synthetic asset
function run() {asset("#FLAT"); // Dummy assets
```

```
// Set all prices to 50for(int i = 0; i < LookBack; i++) {priceSet(0, 50, -i); // OpenpriceSet(1, 50, -i); // HighpriceSet(2, 50, -i); // LowpriceSet(3, 50, -i); // Close}}

// Basket trading - create "USD" assets combinedvar priceUSD() {
var p = 0;
assets("GBP/USD"); p += price();asset("AUD/USD"); p += price();asset("EUR/USD"); p += price();return p;}

function run() {asset("#USD"); // Synthetic asset

// Set close to USD value basketpriceSet(3, priceUSD(), 0);

// Trading on basketvars Price = series(priceClose());if(crossOver(Price, SMA(Price, 20)))enterLong();}

// Tick simulation for HFT backtestfunction run() {for(int i = 0; i < 100; i++) {// Simulate tick pricevar tick = priceLow(0) +random() * (priceHigh(0) - priceLow(0));priceSet(4, tick, 0); // Set tick

// Strategy based on tickif(priceQuote() > threshold)enterLong();}}
// Create flat synthetic asset
function run() {
assets("#FLAT"); // Dummy assets
// Set all prices to 50
for(int i = 0; i < LookBack; i++) {
priceSet(0, 50, -i); // Open
priceSet(1, 50, -i); // High
priceSet(2, 50, -i); // Low
priceSet(3, 50, -i); // Close
}
}
// Basket trading - create combined "USD" assets
var priceUSD() {
var p = 0;
assets("GBP/USD"); p += price();
assets("AUD/USD"); p += price();
assets("EUR/USD"); p += price();
return p;
}
function run() {
assets("#USD"); // Synthetic asset
// Set close to USD basket value
```

```
priceSet(3, priceUSD(), 0);
// Basketball Trading
vars Price = series(priceClose());
if(crossOver(Price, SMA(Price, 20)))
enterLong();
}
// Tick simulation for HFT backtest
function run() {
for(int i = 0; i < 100; i++) {
// Simulate tick price
var tick = priceLow(0) +
random() * (priceHigh(0) - priceLow(0));
priceSet(4, tick, 0); // Set tick
// Tick-based strategy
if(priceQuote() > threshold)
enterLong();
}
}
```

Note:\

- Temporary modification (in memory only)\
- Does not save to History file\
- Useful for synthetic assets and simulations\
- priceSet(4, ...) feeds priceQuote()\
- Negative offset writes into the future (test mode only)

---

priceReal - Price Without Spread

Description: Returns the "real" price with no spread applied. In backtests with bid/ask data, returns the mid price. Useful for precise P&L calculations.

Syntax:

Note:

- Temporary change (in memory only)
- Does not save to History file
- Useful for synthetic assets and simulations
- priceSet(4, ...) feeds priceQuote()
- Negative offset writes to the future (test mode only)

2.3.4. priceReal - Spread-Free Price

Description:

Returns the "real" price without the spread applied. When backtesting with bid/ask data, it returns the mid-price. Useful for precise P&L calculations.

Syntax:

var priceReal(int Offset);

var priceReal(int Offset);

Parameters:\

- Offset (int): Number of bars back (0 = current)

  Return Value:var - Actual price (mid) at the specified bar

  Example:

Parameters:

- **Offset** (int): Number of bars back (0 = current)

Return Value:

var - Actual price (mid) at the specified bar

Example:

function run() {
// Price with spread included (what you see)var priceWithSpread = priceClose(0);

// Real price without spreadvar realPrice = priceReal(0);

// Spread applied
var spread = priceWithSpread - realPrice;

printf("\nPrice: %.5f, Real: %.5f, Spread: %.5f",priceWithSpread, realPrice, spread);

// Precise P&L calculationvar entryReal = priceReal(0);enterLong();// ... after a while...var exitReal = priceReal(0);var truePL = (exitReal - entryReal) * Lots * PIP/PIPCost;}

function run() {

// Price with spread included (what you see)

var priceWithSpread = priceClose(0);

// Real price without spread

var realPrice = priceReal(0);

// Spread applied

var spread = priceWithSpread - realPrice;

printf("\nPrice: %.5f, Real: %.5f, Spread: %.5f",

priceWithSpread, realPrice, spread);

```
// Accurate P&L calculation
var entryReal = priceReal(0);
enterLong();
// ... after a while ...
var exitReal = priceReal(0);
var truePL = (exitReal - entryReal) * Lots * PIP/PIPCost;
}
```

Note:\

- With .t1 data (bid/ask): returns (bid+ask)/2\
- With .t6 data: returns priceClose\
- Useful for theoretical calculations without slippage\
- Do not use for real-time entry/exit decisions.

---

priceRecord - Save Prices to History

Description: Saves current prices to the History file. Used to record data from custom sources or to save synthetic assets.

Syntax:

Note:

- With data .t1 (bid/ask): returns (bid+ask)/2
- With .t6 data: returns priceClose
- Useful for theoretical calculations without slippage
- Do not use for real-time entry/exit decisions

2.3.5. priceRecord - Save Prices in History

Description:

Saves current prices to the History file. Used to record data from custom sources or to save synthetic assets.

Syntax:

```
int priceRecord();
int priceRecord();
```

Return Value:
int - Number of records saved, 0 if error

Example:

Return Value:

int - Number of records saved, 0 if error

Example:

```
// Record basket prices
function run() {asset("#USDBASKET");

// Calculate price basketvar basketPrice = 0;asset("EUR/USD"); basketPrice +=
priceClose(0);asset("GBP/USD"); basketPrice += priceClose(0);asset("AUD/USD"); basketPrice +=
priceClose(0);

// Set and saveasset("#USDBASKET");priceSet(3, basketPrice, 0);priceRecord(); // Save to
History/USDBASKET.t6}

// Save data from API customfunction tick() {
// Get data from external API
var tickPrice = getExternalAPIPrice();

asset("CUSTOMFEED");priceSet(4, tickPrice, 0); // Set tickpriceRecord(); // Save}
// Record basket prices
function run() {
asset("#USDBASKET");
// Calculate basket price
var basketPrice = 0;
assets("EUR/USD"); basketPrice += priceClose(0);
assets("GBP/USD"); basketPrice += priceClose(0);
assets("AUD/USD"); basketPrice += priceClose(0);
// Set and save
asset("#USDBASKET");
priceSet(3, basketPrice, 0);
priceRecord(); // Save to History/USDBASKET.t6
}
// Save data from custom API
function tick() {
// Get data from external API
var tickPrice = getExternalAPIPrice();
asset("CUSTOMFEED");
priceSet(4, tickPrice, 0); // Set tick
priceRecord(); // Save
}
```

Note:\

- Save to History/AssetNameYear.t6\

- Create file if it does not exist\
- Append to existing data\
- Does not duplicate existing timestamps\
- Requires all OHLC prices to be set

MARKET DATA

These functions return additional market data in addition to OHLC prices.

marketVal - Spread or Market Value

Description: Returns additional market data: bid-ask spread, quote size, or custom field fVal from a .t6 file.

Syntax:

Note:

- Save to History/AssetNameYear.t6
- Create file if it does not exist
- Append to existing data
- Does not duplicate existing timestamps
- Requires all OHLC prices to be set

2.4. MARKET DATA

These functions return additional market data in addition to OHLC prices.

2.4.1. marketVal - Spread or Market Value

Description:

Returns additional market data: bid-ask spread, quote size, or custom field fVal from .t6 files.

Syntax:

var marketVal(int Offset);

var marketVal(int Offset);

Parameters:\

- Offset (int): Number of bars back (0 = current)

Return Value:var - Depends on available data:\

- LIVE: current bid-ask spread\
- .t1/.t2: spread from the bar\

- .t6: fVal field

  Example:

Parameters:

- **Offset** (int): Number of bars back (0 = current)

Return Value:

var - Depends on available data:

- LIVE: Current bid-ask spread
- .t1/.t2: spread from the bar
- .t6: fVal field

Example:

```
function run() {
// Use variable spread from historical data if(is(TEST))Spread = max(0, marketVal(0));

// Tight spread trading if(marketVal(0) < 0.0002) // Spread < 2 pipsenterLong();}

// In TMF or tick functionfunction tick() {var spread = marketVal(0); // Spread real-timeprintf("\nCurrent spread: %.5f", spread);}
function run() {
// Use variable spread from historical data
if(is(TEST))
Spread = max(0, marketVal(0));
// Trading on a narrow spread
if(marketVal(0) < 0.0002) // Spread < 2 pips
enterLong();
}
// In TMF or tick function
function tick() {
var spread = marketVal(0); // Real-time spread
printf("\nCurrent spread: %.5f", spread);
}
```

Note:\

- Requires Zorro S and no LEAN flag\
- In LIVE it depends on the API broker\
- In .t6: store custom value in fVal field\
- marketVal(0) in tick() = spread last tick

  ---

  marketVol - Volume or Tick Frequency

Description:
Returns the trading volume or proxy (tick frequency, quote size) of the current bar.

Syntax:

Note:

- Requires Zorro S and no LEAN flag
- In LIVE it depends on the API broker
- In .t6: store custom value in fVal field
- marketVal(0) in tick() = last tick spread

2.4.2. marketVol - Volume or Tick Frequency

Description:

Returns the trading volume or proxy (tick frequency, quote size) of the current bar.

Syntax:

var marketVol(int Offset);

var marketVol(int Offset);

Parameters:\

- Offset (int): Number of bars back (0 = current)

Return Value:var - Depends on data:\

- LIVE: accumulated volume or tick frequency\
- .t1: sum of quote sizes\
- .t6: fVol field\
- 0 if not available

Example:

Parameters:

- **Offset** (int): Number of bars back (0 = current)

Return Value:

var - Depends on the data:

- LIVE: accumulated volume or tick frequency
- .t1: sum of quote sizes
- .t6: fVol field
- 0 if not available

Example:

```
function run() {
// Volume-based entryvar avgVol = SMA(series(marketVol()), 20);
```

```
if(marketVol(0) > 2.0 * avgVol) // Volume spike
if(priceClose(0) > priceClose(1))enterLong(); // Breakout with volume

// Volume total per timeframevar totalVol = marketVol(0) * BarPeriod / 1; // If given M1}

// Order flow analysisfunction run() {BarPeriod = 1; // M1 bars

vars Vol = series(marketVol());vars Price = series(priceClose());

// Price-volume correlationif(Vol[0] > Vol[1] && Price[0] > Price[1])printf("\nBullish volume");}
function run() {
// Volume-based entry
var avgVol = SMA(series(marketVol()), 20);
if(marketVol(0) > 2.0 * avgVol) // Volume spike
if(priceClose(0) > priceClose(1))
enterLong(); // Breakout with volume
// Total volume per timeframe
var totalVol = marketVol(0) * BarPeriod / 1; // If given M1
}
// Order flow analysis
function run() {
BarPeriod = 1; // M1 bars
vars Vol = series(marketVol());
vars Price = series(priceClose());
// Price-volume correlation
if(Vol[0] > Vol[1] && Price[0] > Price[1])
printf("\nBullish volume");
}
```

Note:\

- Requires Zorro S (except if in .t6)\
- Volume type depends on the broker\
- Some brokers: tick frequency instead of volume\
- SETVOLTYPE to change volume source\
- LIVE Volume ≠ Historical Volume (different filters)

ADVANCED MARKET DATA

dayOpen, dayHigh, dayLow, dayClose - Daily Prices

Description: Returns OHLC prices for the current or specified trading day, regardless of the bars' timeframe.

Syntax:

Note:

- Requires Zorro S (except if in .t6)
- Volume type depends on the broker
- Some brokers: tick frequency instead of volume
- SETVOLTYPE to change volume source
- LIVE volume ≠ historical volume (different filters)

2.5. ADVANCED MARKET DATA

2.5.1. dayOpen, dayHigh, dayLow, dayClose - Daily Prices

Description:

Returns OHLC prices for the current or specified trading day, regardless of the bars' timeframe.

Syntax:

var dayOpen(int Offset);
var dayHigh(int Offset);var dayLow(int Offset);var dayClose(int Offset);

var dayOpen(int Offset);

var dayHigh(int Offset);

var dayLow(int Offset);

var dayClose(int Offset);

Parameters:\

- Offset (int): Number of days back (0 = today, 1 = yesterday, etc.)

Return Value:
var - Requested daily price

Example:

Parameters:

- **Offset** (int): Number of days back (0 = today, 1 = yesterday, etc.)

Return Value:

var - Daily asking price

Example:

// Intraday trading based on daily range
function run() {BarPeriod = 5; // 5 minutes

var todayOpen = dayOpen(0);var todayHigh = dayHigh(0);var todayLow = dayLow(0);

```
// Range breakoutif(priceClose(0) > todayHigh)enterLong();else if(priceClose(0) < todayLow)enterShort();

// Mean reversion from day extremesvar dayRange = todayHigh - todayLow;var midDay = todayLow + dayRange/2;

if(priceClose(0) > todayHigh - 0.2*dayRange) // Near highenterShort(); // Fade the moveelse if(priceClose(0) < todayLow + 0.2*dayRange) // Near lowenterLong();}

// Comparison with yesterdayfunction run() {if(dayClose(0) > dayClose(1)) // Today > yesterdayprintf("\nBullish day");}
// Intraday trading based on daily range
function run() {
BarPeriod = 5; // 5 minutes
var todayOpen = dayOpen(0);
var todayHigh = dayHigh(0);
var todayLow = dayLow(0);
// Range breakout
if(priceClose(0) > todayHigh)
enterLong();
else if(priceClose(0) < todayLow)
enterShort();
// Mean reversion from day extremes
var dayRange = todayHigh - todayLow;
var midDay = todayLow + dayRange/2;
if(priceClose(0) > todayHigh - 0.2*dayRange) // Near high
enterShort(); // Fade the move
else if(priceClose(0) < todayLow + 0.2*dayRange) // Near low
enterLong();
}
// Comparison with yesterday
function run() {
if(dayClose(0) > dayClose(1)) // Today > yesterday
printf("\nBullish day");
}
```

Note:\

- Works with any BarPeriod\
- Based on BarZone (bar timezone)\
- dayClose(0) = last close of the day (can be future if intraday)\

- Useful for intraday strategies with daily references

---

dayPivot - Daily Pivot Points

Description: Calculates daily pivot points (Pivot, R1, R2, R3, S1, S2, S3) based on the previous day's prices.

Syntax:

Note:

- Works with any BarPeriod
- Based on BarZone (bar timezone)
- dayClose(0) = last close of the day (can be future if intraday)
- Useful for intraday strategies with daily references

2.5.2. dayPivot - Daily Pivot Points

Description:

Calculates daily pivot points (Pivot, R1, R2, R3, S1, S2, S3) based on the previous day's prices.

Syntax:

var dayPivot(int Type, int Offset);

var dayPivot(int Type, int Offset);

Parameters:\

- Type (int): Pivot type\
- 0 = Pivot Point (P)\
- 1 = Resistance 1 (R1)\
- 2 = Resistance 2 (R2)\
- 3 = Resistance 3 (R3)\
- -1 = Support 1 (S1)\
- -2 = Support 2 (S2)\
- -3 = Support 3 (S3)\
- Offset (int): Days back (0 = today)

Return Value:var - Calculated pivot level

Example:

Parameters:

- **Type** (int): Pivot type

- 0 = Pivot Point (P)

- 1 = Resistance 1 (R1)
- 2 = Resistance 2 (R2)
- 3 = Resistance 3 (R3)
- -1 = Support 1 (S1)
- -2 = Support 2 (S2)
- -3 = Support 3 (S3)

- **Offset** (int): Days back (0 = today)

Return Value:

var - Calculated Pivot Level

Example:

```
// Trading with pivot points
function run() {BarPeriod = 15; // 15 minutes

var P = dayPivot(0, 0); // Pivot
var R1 = dayPivot(1, 0); // Resistance 1var S1 = dayPivot(-1, 0); // Support 1

// Long near supportif(between(priceClose(0), S1 - 5*PIP, S1 + 5*PIP))enterLong();

// Short near resistanceif(between(priceClose(0), R1 - 5*PIP, R1 + 5*PIP))enterShort();

// Plot pivot levelsplot("Pivot", P, LINE, CYAN);plot("R1", R1, LINE, RED);plot("S1", S1, LINE, GREEN);}

// Manual calculation pivotvar pivot = (dayHigh(1) + dayLow(1) + dayClose(1)) / 3;var r1 = 2*pivot - dayLow(1);var s1 = 2*pivot - dayHigh(1);
// Trading with pivot points
function run() {
BarPeriod = 15; // 15 minutes
var P = dayPivot(0, 0); // Pivot
var R1 = dayPivot(1, 0); // Resistance 1
var S1 = dayPivot(-1, 0); // Support 1
// Long near support
if(between(priceClose(0), S1 - 5*PIP, S1 + 5*PIP))
enterLong();
// Short near resistance
if(between(priceClose(0), R1 - 5*PIP, R1 + 5*PIP))
enterShort();
// Pivot-level plot
plot("Pivot", P, LINE, CYAN);
plot("R1", R1, LINE, RED);
plot("S1", S1, LINE, GREEN);
```

```
}
// Manual pivot calculation
var pivot = (dayHigh(1) + dayLow(1) + dayClose(1)) / 3;
var r1 = 2*pivot - dayLow(1);
var s1 = 2*pivot - dayHigh(1);
```

Note:\

- Based on: P = (H + L + C) / 3 of the previous day\
- R1 = 2P - L, S1 = 2P - H\
- R2 = P + (H - L), S2 = P - (H - L)\
- R3 = H + 2(P - L), S3 = L - 2(H - P)\
- Updated every day\
- Widely used in day trading

POSITION MANAGEMENTenterLong / enterShort - Opening PositionsDescription: Opens a long (buy) or short (sell) position on the current asset. The size is determined by Lots or by automatic calculation based on Capital/Risk.Syntax:

Note:

- Based on: P = (H + L + C) / 3 of the previous day
- R1 = 2P - L, S1 = 2P - H
- R2 = P + (H - L), S2 = P - (H - L)
- R3 = H + 2(P - L), S3 = L - 2(H - P)
- Updated every day
- Widely used in day trading

2.6. POSITION MANAGEMENT

2.6.1. enterLong / enterShort - Opening Positions

Description:

Opens a long (buy) or short (sell) position on the current asset. The size is determined by Lots or by automatic calculation based on Capital/Risk.

Syntax:

```
TRADE* enterLong(int NumLots, var Limit, var Stop, var TakeProfit);
TRADE* enterShort(int NumLots, var Limit, var Stop, var TakeProfit);

// Alternative versionsTRADE* enterLong(function TMF, var v0, ...); // With TMF
TRADE* enterLong(int Lots); // Lots only
TRADE* enterLong(); // All automatic
```

TRADE* enterLong(int NumLots, var Limit, var Stop, var TakeProfit);

TRADE* enterShort(int NumLots, var Limit, var Stop, var TakeProfit);

// Alternative versions

TRADE* enterLong(function TMF, var v0, ...); // With TMF

TRADE* enterLong(int Lots); // Lots only

TRADE* enterLong(); // All automatic

Parameters:\

- NumLots (int): Number of contracts (0 = use global Lots or automatic calculation)
- Limit (var): Limit price per entry (0 = market order)\
- Stop (var): Stop loss in pips (0 = use Global Stop)\
- TakeProfit (var): Take profit in pips (0 = use global TakeProfit)\
- TMF (function): Trade Management Function\
- v0-v7 (var): Parameters for TMF

  Return Value: TRADE* - Pointer to the open trade, NULL if failed

  Example:

Parameters:

- **NumLots** (int): Number of contracts (0 = use global Lots or automatic calculation)
- **Limit** (var): Limit price per entry (0 = market order)
- **Stop** (var): Stop loss in pips (0 = use Global Stop)
- **TakeProfit** (var): Take profit in pips (0 = use global TakeProfit)
- **TMF** (function): Trade Management Function
- **v0-v7** (var): Parameters for TMF

Return Value:

TRADE* - Pointer to the open trade, NULL if failed

Example:

```
// Market order - 1 lot
enterLong(1);

// Market order - autosizeLots = 10;enterLong();

// Limit order with stop and targetenterLong(2, priceClose() * 0.99, 50, 100);// | | | |
// | limit -1% | 50 pips stop
// 2 contracts +100 pips profit

// Automatic position sizing based on Riskfunction run() {BarPeriod = 60;asset("EUR/USD");

var atr = ATR(14);Stop = 2 * atr; // Stop at 2 ATRRisk = 0.02; // Risk 2% of capital

// Zorro automatically calculates LotsenterLong();}
```

// With Trade Management Functionfunction myTMF(var v0) {if(TradeProfit > v0)exitTrade(); // Exit on targetreturn 0;}

enterLong(myTMF, 100); // Exit when profit > 100
// Market order - 1 lot
enterLong(1);
// Market order - automatic size
Lots = 10;
enterLong();
// Limit order with stop and target
enterLong(2, priceClose() * 0.99, 50, 100);
// | | | |
// | limit -1% | 50 pip stop
// 2 contracts +100 pips profit
// Automatic position sizing based on risk
function run() {
BarPeriod = 60;
asset("EUR/USD");
var atr = ATR(14);
Stop = 2 * atr; // Stop at 2 ATR
Risk = 0.02; // Risk 2% of capital
// Zorro automatically calculates Lots
enterLong();
}
// With Trade Management Function
function myTMF(var v0) {
if(TradeProfit > v0)
exitTrade(); // Exit on target
return 0;
}
enterLong(myTMF, 100); // Exit when profit > 100
Note:\

- Stop and TakeProfit are in pips, not absolute price.
- NumLots = 0 uses automatic Risk-based calculation\
- Check NumPendingTotal for pending orders\
- In Hedge=2 positions are not closed automatically\
- For options: use contract() + enterLong/Short()

---

exitLong / exitShort - Close Positions

Description: Closes all open long or short positions on the current asset. To close specific positions, use exitTrade().

Syntax:

Note:

- Stop and TakeProfit are in pips, not absolute price
- NumLots = 0 uses automatic risk-based calculation
- Check NumPendingTotal for pending orders
- In Hedge=2 positions are not closed automatically
- For options: use contract() + enterLong/Short()

2.6.2. exitLong / exitShort - Closing Positions

Description:

Closes all open long or short positions on the current asset. To close specific positions, use exitTrade().

Syntax:

int exitLong(var Limit, var NumLots);
int exitShort(var Limit, var NumLots);

// Simplified versions
int exitLong(); // Close everything to mercatoint exitShort();

int exitLong(var Limit, var NumLots);

int exitShort(var Limit, var NumLots);

// Simplified versions

int exitLong(); // Close all market

int exitShort();

Parameters:\

- Limit (var): Limit price for exit (0 = market order)\
- NumLots (var): Number of lots to close (0 = close all)

Return Value: int - Number of closed positions

Example:

Parameters:

- **Limit** (var): Limit price for exit (0 = market order)
- **NumLots** (var): Number of lots to close (0 = close all)

Return Value:

int - Number of closed positions

Example:

```
// Close all longs at market
exitLong();

// Close all shorts exitShort();

// Scale out - close partially if(NumOpenLong > 0) {var profit = TradeProfit; if(profit > 100) exitLong(0, NumOpenLong / 2); // Close half}

// Exit with limit exitLong(priceClose() * 1.01); // Sell at +1%

// Position Management function run() { // Open
if(crossOver(Price, SMA))
enterLong();

// Close on opposite signal if(crossUnder(Price, SMA)) exitLong();

// Manual stop loss if(TradeProfit < -100) exitLong();}
// Close all long positions on the market
exitLong();
// Close all shorts
exitShort();
// Scale out - partially close
if(NumOpenLong > 0) {
var profit = TradeProfit;
if(profit > 100)
exitLong(0, NumOpenLong / 2); // Close half
}
// Exit with limit
exitLong(priceClose() * 1.01); // Sell at +1%
// Position Management
function run() {
// Open
if(crossOver(Price, SMA))
enterLong();
// Close on opposite signal
if(crossUnder(Price, SMA))
exitLong();
```

```
// Manual stop loss
if(TradeProfit < -100)
exitLong();
}
```

Note:\

- exitLong() closes ONLY long, not short.
- In Hedge=2 you can be long and short at the same time.
- Limit orders may not execute immediately\
- To close everything: loop with exitTrade()

---

exitTrade - Close Specific Trade

Description: Closes a specific trade identified by the TRADE* pointer.

Syntax:

Note:

- exitLong() closes ONLY long, not short
- In Hedge=2 you can be long and short at the same time
- Limit orders may not execute immediately
- To close everything: loop with exitTrade()

2.6.3. exitTrade - Close Specific Trade

Description:

Closes a specific trade identified by the TRADE* pointer.

Syntax:

```
void exitTrade(TRADE* tr);
void exitTrade(TRADE* tr, var Limit);
void exitTrade(TRADE* tr);
void exitTrade(TRADE* tr, var Limit);
```

Parameters:\

- tr(TRADE*): Pointer to the trade to close (NULL = current trade in the loop)\
- Limit (var): Limit price (0 = market)

Return Value: void

Example:

Parameters:

- **tr** (TRADE*): Pointer to the trade to close (NULL = current trade in the loop)

- **Limit** (var): Limit price (0 = market)

Return Value:

void

Example:

```
// Close first open trade
for(opentrades) {
exitTrade(); // Close the current trade of the loop
break;}

// Close specific trade TRADE* myTrade = enterLong();// ... after a while ... exitTrade(myTrade);

// Close oldest trade TRADE* oldestTrade = NULL;var oldestTime = 0; for(opentrades) {
if(TradeOpenBar > oldestTime) { oldestTime = TradeOpenBar; oldestTrade = ThisTrade;}}
if(oldestTrade) exitTrade(oldestTrade);

// Close only losing trades for(opentrades) { if(TradeProfit < -50) exitTrade();}
// Close first open trade
for(opentrades) {
exitTrade(); // Closes the current trade in the loop
break;
}
// Close specific trade
TRADE* myTrade = enterLong();
// ... after a while ...
exitTrade(myTrade);
// Close older trade
TRADE* oldestTrade = NULL;
var oldestTime = 0;
for(opentrades) {
if(TradeOpenBar > oldestTime) {
oldestTime = TradeOpenBar;
oldestTrade = ThisTrade;
}
}
if(oldestTrade)
exitTrade(oldestTrade);
// Close only losing trades
for(opentrades) {
if(TradeProfit < -50)
```

exitTrade();

}

Note:\

- NULL closes the current trade in the for(opentrades) loop.
- Limit order: may not execute immediately\
- Trade pointer valid until close\
- After exit, the pointer is no longer valid

OPTIONS & FUTURES

contract - Contract Selection

Description: Select an option or futures contract from the chain loaded with contractUpdate(). The contract is identified by type, strike, and expiration.

Syntax:

Note:

- NULL closes the current trade in the for(opentrades) loop
- Limit order: may not execute immediately
- Trade pointer valid until close
- After exit, the pointer is no longer valid

2.7. OPTIONS & FUTURES

2.7.1. contract - Contract Selection

Description:

Select an option or futures contract from the loaded chain with contractUpdate(). The contract is identified by type, strike, and expiration.

Syntax:

bool contract(int Type);
bool contract(int Type, var Strike);bool contract(int Type, var Strike, var Days);

bool contract(int Type);

bool contract(int Type, var Strike);

bool contract(int Type, var Strike, var Days);

Parameters:\

- Type (int): Contract type\
- CALL = call option\
- PUT = put option\

- FUTURE = futures contract\
- Strike (var): Strike price (options) or 0 (futures)\
- Days (var): Days to expiration (DTE) or absolute date (YYYYMMDD)

Return Value: bool - TRUE if contract found, FALSE otherwise

Example:

Parameters:

- **Type** (int): Contract type
  - CALL = call option
  - PUT = put option
  - FUTURE = futures contract
- **Strike** (var): Strike price (options) or 0 (futures)
- **Days** (var): Days to Expiration (DTE) or absolute date (YYYYMMDD)

Return Value:

bool - TRUE if contract found, FALSE otherwise

Example:

```
function run() {
Hedge = 2; // Mode optionsasset("SPX");

// Load option chainif(!contractUpdate(Asset, 0, CALL|PUT)) {printf("\nERROR: Failed to load chain");return;}

// Select CALL strike 4500, ~30 DTEif(contract(CALL, 4500, 30)) {printf("\nContract: %s", ContractID);
printf("\nPrice: %.2f", ContractPrice);
printf("\nDelta: %.3f", ContractDelta);printf("\nDTE: %.0f", contractDays());

enterShort(); // Sell CALL}

// Automatic ATM strikevar atmStrike = priceClose();atmStrike = floor(atmStrike / 5) * 5; // Round to $5

if(contract(PUT, atmStrike, 45))enterLong(); // Buy PUT}

// Short Straddlefunction run() {var strike = priceClose();var expiry = 7; // 7 DTE

contract(CALL, strike, expiry);enterShort(1);

contract(PUT, strike, expiry);enterShort(1);}
function run() {
Hedge = 2; // Options mode
asset("SPX");
```

```
// Load option chain
if(!contractUpdate(Asset, 0, CALL|PUT)) {
printf("\nERROR: Failed to load chain");
return;
}
// Select CALL strike 4500, ~30 DTE
if(contract(CALL, 4500, 30)) {
printf("\nContract: %s", ContractID);
printf("\nPrice: %.2f", ContractPrice);
printf("\nDelta: %.3f", ContractDelta);
printf("\nDTE: %.0f", contractDays());
enterShort(); // Sell CALL
}
// Automatic ATM strike
var atmStrike = priceClose();
atmStrike = floor(atmStrike / 5) * 5; // Round to $5
if(contract(PUT, atmStrike, 45))
enterLong(); // Buy PUT
}
// Short Straddle
function run() {
var strike = priceClose();
var expiry = 7; // 7 DTE
contract(CALL, strike, expiry);
enterShort(1);
contract(PUT, strike, expiry);
enterShort(1);
}
```

Note:\

- Requires Hedge = 2\
- contractUpdate() must be called first\
- Strike rounded to the nearest available value\
- Days: Relative DTE or absolute date YYYYMMDD\
- Available variables: ContractPrice, ContractDelta, ContractGamma, etc.

---

contractUpdate - Option Chain Update

Description: Downloads and updates the options/futures chain from the broker or historical data. Required before using contract().

Syntax:

Note:

- Requires Hedge = 2
- contractUpdate() must be called first
- Strike rounded to the nearest available value
- Days: Relative DTE or absolute date YYYYMMDD
- Available variables: ContractPrice, ContractDelta, ContractGamma, etc.

2.7.2. contractUpdate - Option Chain Update

Description:

Download and update the options/futures chain from your broker or historical data. Required before using contract().

Syntax:

int contractUpdate(string Underlying, int Type, int Filter);

int contractUpdate(string Underlying, int Type, int Filter);

Parameters:\

- Underlying (string): Symbol of the underlying (e.g. "SPX", "SPY", "ES")\
- Type (int): FUTURE for futures, 0 for options\
- Filter (int): Filter by type\
- CALL = call only\
- PUT = put only\
- CALL|PUT = call and put\
- 0 = all

Return Value:int - Number of contracts loaded, 0 if failed

Example:

Parameters:

- **Underlying** (string): Symbol of the underlying (e.g. "SPX", "SPY", "ES")
- **Type** (int): FUTURE for futures, 0 for options
- **Filter** (int): Filter by type
  - CALL = call only
  - PUT = put only
  - CALL|PUT = call and put
  - 0 = all

Return Value:

int - Number of contracts loaded, 0 if failed

Example:

```
function run() {
Hedge = 2;asset("SPX");

// Load all optionsint num = contractUpdate("SPX", 0, CALL|PUT);if(num == 0) {printf("\nERROR: No contracts");quit();}
printf("\nLoaded %d contracts", num);

// Only CALLcontractUpdate("SPY", 0, CALL);

// FuturescontractUpdate("ES", FUTURE, 0);}

// Iterate over the chainfunction run() {contractUpdate("SPX", 0, CALL|PUT);

CONTRACT* C;while(C = contractNext(0)) {printf("\n%s Strike:%.0f DTE:%.0f Delta:%.3f",ContractType == CALL ? "CALL" : "PUT",ContractStrike,contractDays(),ContractDelta);}}
function run() {
Hedge = 2;
asset("SPX");
// Load all options
int num = contractUpdate("SPX", 0, CALL|PUT);
if(num == 0) {
printf("\nERROR: No contracts");
quit();
}
printf("\nLoaded %d contracts", num);
// CALL only
contractUpdate("SPY", 0, CALL);
// Futures
contractUpdate("ES", FUTURE, 0);
}
// Iterate over the chain
function run() {
contractUpdate("SPX", 0, CALL|PUT);
CONTRACT* C;
while(C = contractNext(0)) {
printf("\n%s Strike:%.0f DTE:%.0f Delta:%.3f",
ContractType == CALL ? "CALL" : "PUT",
```

ContractStrike,
contractDays(),
ContractDelta);
}
}
Note:\

- Slow: ~20 minutes with some brokers\
- In backtest: load contracts available at the given bar\
- LIVE: Call once a day, preferably after hours.
- Prices update automatically for open trades.
- In test/train: call at each bar to update prices\
- Some brokers return contracts that have already expired - check expiry

---

contractDays - Days to Expiration

Description: Returns the days remaining until expiration (DTE - Days To Expiration) of the current contract.

Syntax:

Note:

- Slow: ~20 minutes with some brokers
- In backtest: load contracts available at the given bar
- LIVE: Call once a day, preferably after hours.
- Prices update automatically for open trades
- In test/train: call at each bar to update prices
- Some brokers return expired contracts - check expiry

2.7.3. contractDays - Days to Expiration

Description:

Returns the days remaining until expiration (DTE - Days To Expiration) of the current contract.

Syntax:

var contractDays();
var contractDays(var Expiry); // From specific date
var contractDays();
var contractDays(var Expiry); // From specific date
Parameters:\

- Expiry (var): [Optional] Expiry date YYYYMMDD

  Return Value:var - Days to expiry (decimal)

  Example:

Parameters:

- **Expiry** (var): [Optional] Expiry date YYYYMMDD

Return Value:

var - Days to Expiration (decimal)

Example:

```
// DTE of the selected
contract contract(CALL, 4500, 30);var dte = contractDays();printf("\nDays to expiry: %.1f", dte);

// Filter by DTEif(contractDays() > 45 && contractDays() < 55) {// Trade only ~50 DTEenterShort();}

// Days from specified datevar daysTo = contractDays(20250315); // As of 15 Mar 2025
// DTE of the selected contract
contract(CALL, 4500, 30);
var dte = contractDays();
printf("\nDays to expire: %.1f", dte);
// Filter by DTE
if(contractDays() > 45 && contractDays() < 55) {
// Trade only ~50 DTE
enterShort();
}
// Days from specific date
var daysTo = contractDays(20250315); // To 15 Mar 2025
```

Note:\

- Counts calendar days, not trading days.
- Decimal for intraday precision\
- Use ContractExpiry (OLE data) for calculations\
- ContractHour determines expiration time (default 1200 UTC)

  ---

  contractExpiry - Expiry Date

  Description: Finds the expiration date closest to a target number of days from the loaded chain.

  Syntax:

Note:

- Count calendar days, not trading days
- Decimal for intraday precision
- Use ContractExpiry (OLE data) for calculations
- ContractHour determines the expiration time (default 1200 UTC)

2.7.4. contractExpiry - Expiry Date

Description:

Find the closest expiration date to a target number of days from the uploaded chain.

Syntax:

var contractExpiry(var Days);

var contractExpiry(var Days);

Parameters:\

- Days (var): DTE target

  Return Value:var - Expiration date in OLE format

  Example:

Parameters:

- **Days** (var): DTE target

Return Value:

var - Expiration date in OLE format

Example:

```
// Find expiry ~45 days
var expiry45 = contractExpiry(45);

// Use for comboLegvar strike = priceClose();comboLeg(strike, CALL, expiry45);comboLeg(strike, PUT, expiry45);
// Find expiration ~45 days
var expiry45 = contractExpiry(45);
// Use for comboLeg
var strike = priceClose();
comboLeg(strike, CALL, expiry45);
comboLeg(strike, PUT, expiry45);
```

Note:\

- Return to nearest available expiration\
- OLE Format (days from 12/30/1899)\
- Use with comboLeg for multi-leg strategies

---

contractNext - Contract Iteration

Description: Returns the next contract in the chain to iterate over all available contracts.

Syntax:

Note:

- Return to nearest available expiration
- OLE Format (days from 12/30/1899)
- Use with comboLeg for multi-leg strategies

2.7.5. contractNext - Contract Iteration

Description:

Returns the next contract in the chain to iterate over all available contracts.

Syntax:

CONTRACT* contractNext(int Mode);

CONTRACT* contractNext(int Mode);

Parameters:\

- Mode (int):\
- 0 = next in chain\
- CALL = call only\
- PUT = put only

Return Value:CONTRACT* - Pointer to the contract, NULL if finished

Example:

Parameters:

- **Mode** (int):

- 0 = next in the chain
- CALL = call only
- PUT = put only

Return Value:

CONTRACT* - Pointer to the contract, NULL if finished

Example:

```
// Iterate over all contracts
CONTRACT* C;while(C = contractNext(0)) {printf("\n%s Strike:%.0f DTE:%.0f Bid:%.2f",ContractType
== CALL ? "CALL" : "PUT",
ContractStrike,contractDays(),ContractBid);}
```

// Only CALLwhile(contractNext(CALL)) {if(ContractDelta > 0.30 && ContractDelta < 0.35) {// Found ~30 delta CALLenterShort();break;}}

// Find the most liquid contractvar maxVolume = 0;CONTRACT* bestContract = NULL;while(contractNext(0)) {if(ContractVol > maxVolume) {maxVolume = ContractVol;bestContract = ThisContract;}}

```
// Iterate over all contracts
CONTRACT* C;
while(C = contractNext(0)) {
printf("\n%s Strike:%.0f DTE:%.0f Bid:%.2f",
ContractType == CALL ? "CALL" : "PUT",
ContractStrike,
contractDays(),
ContractBid);
}
// CALL only
while(contractNext(CALL)) {
if(ContractDelta > 0.30 && ContractDelta < 0.35) {
// Found ~30 delta CALL
enterShort();
break;
}
}
// Find the most liquid contract
var maxVolume = 0;
CONTRACT* bestContract = NULL;
while(contractNext(0)) {
if(ContractVol > maxVolume) {
maxVolume = ContractVol;
bestContract = ThisContract;
}
}
```

Note:\

- First call: Starts from the beginning of the chain\
- Subsequent calls: continues\
- NULL when chain finished\
- Mode filter by contract type

ADVANCED CONTRACT

contractFind - Contract Search by Criteria

Description: Searches the chain for a contract that matches specific criteria (strike, DTE, delta, premium).

Syntax:

Note:

- First call: Starts from the beginning of the chain
- Subsequent calls: continues
- NULL when chain ends
- Mode filters by contract type

2.8. ADVANCED CONTRACT

2.8.1. contractFind - Contract Search by Criteria

Description:

Search the chain for a contract that matches specific criteria (strike, DTE, delta, premium).

Syntax:

CONTRACT* contractFind(int Type, var Days, var Strike, var Premium);

CONTRACT* contractFind(int Type, var Days, var Strike, var Premium);

Parameters:\

- Type (int): CALL or PUT\
- Days (var): DTE target\
- Strike (var): Strike target or 0 for auto\
- Premium (var): Premium bid/ask target or 0

Return Value:CONTRACT* - Contract found or NULL

Example:

Parameters:

- **Type** (int): CALL or PUT
- **Days** (var): DTE target
- **Strike** (var): Strike target or 0 for auto
- **Premium** (var): Premium bid/ask target or 0

Return Value:

CONTRACT* - Contract found or NULL

Example:

```
// Find PUT 45 DTE with $3 premium
CONTRACT* put = contractFind(PUT, 45, 0, 3.00);if(put) {printf("\nFound: Strike %.0f, Premium %.2f",ContractStrike, ContractBid);}

// Strangle with specific premiumif(combo(
contractFind(CALL, 42, 0, 2.50), 1,
contractFind(PUT, 42, 0, 2.50), 1,0,0,0,0)) {

MarginCost = comboMargin(-1, 3);enterShort(comboLeg(1));enterShort(comboLeg(2));}
// Find PUT 45 DTE with $3 premium
CONTRACT* put = contractFind(PUT, 45, 0, 3.00);
if(put) {
printf("\nFound: Strike %.0f, Premium %.2f",
ContractStrike, ContractBid);
}
// Strangle with specific premium
if(combo(
contractFind(CALL, 42, 0, 2.50), 1,
contractFind(PUT, 42, 0, 2.50), 1,
0,0,0,0)) {
MarginCost = comboMargin(-1, 3);
enterShort(comboLeg(1));
enterShort(comboLeg(2));
}
```

Note:\

- Strike = 0: automatically find the right strike\
- Premium = 0: ignore premium\
- Search for the contract closest to your criteria\
- NULL if no contract satisfies

---

contractCheck - Check Contract Status

Description: Checks whether a contract has been exercised or assigned and manages the resulting underlying position.

Syntax:

Note:

- Strike = 0: automatically finds the right strike

- Premium = 0: ignore premium
- Search for the contract closest to your criteria
- NULL if no contract satisfies

2.8.2. contractCheck - Check Contract Status

Description:

Checks whether a contract has been exercised or assigned and manages the resulting underlying position.

Syntax:

int contractCheck(TRADE* tr, int Mode);

int contractCheck(TRADE* tr, int Mode);

Parameters:\

- tr (TRADE*): Trade to be verified\
- Mode (int):\
- 0 = verify only\
- 1 = sell underlying if present\
- 2 = close trade if exercised

  Return Value:int - Status:\
- 0 = contract still open\
- 1 = exercised\
- 2 = assigned

  Example:

Parameters:

- **tr** (TRADE*): Trade to be verified

- **Mode** (int):

- 0 = verify only
- 1 = sell underlying if present
- 2 = close trade if exercised

Return Value:

int - Status:

- 0 = contract still open
- 1 = exercised
- 2 = assigned

Example:

```
// Check all open trades
for(opentrades) {if(TradeIsOption) {
```

if(contractCheck(ThisTrade, 1)) { // Check and sell underlying
printf("\nContract %s exercised", TradeID);}}}

// Automatic assignment management function run() {// ... trading logic ...

// Daily check
if(hour() == 16) { // After closing for(opentrades) {contractCheck(ThisTrade, 1); // Sell underlying if assigned}}}
// Check all open trades
for(opentrades) {
if(TradeIsOption) {
if(contractCheck(ThisTrade, 1)) { // Check and sell underlying
printf("\nContract %s exercised", TradeID);
}
}
}
// Automatic assignment management
function run() {
// ... trading logic ...
// Daily check
if(hour() == 16) { // After closing
for(opentrades) {
contractCheck(ThisTrade, 1); // Sell underlying if assigned
}
}
}
Note:\

- Requires GETPOSITION from the broker\
- In backtest: sell underlying automatically\
- Mode 1: Sell underlying at market\
- Call after ContractExpiry for European options

---

contractExercise - Option Exercise

Description: Manually exercises an option contract, converting it into an underlying position.

Syntax:

Note:

- Requires GETPOSITION from the broker
- In backtest: sell underlying automatically
- Mode 1: Sell underlying at market
- Call after ContractExpiry for European options

2.8.3. contractExercise - Option Exercise

Description:

Manually exercise an option contract, converting it into an underlying position.

Syntax:

void contractExercise(TRADE* tr, var Price);

void contractExercise(TRADE* tr, var Price);

Parameters:\

- tr (TRADE*): Contract to be exercised\
- Price (var): [Optional] Strike price (0 = strike)

  Return Value: void

  Example:

Parameters:

- **tr** (TRADE*): Contract to be exercised

- **Price** (var): [Optional] Strike price (0 = strike)

Return Value:

void

Example:

```
// Exercise if ITM
for(opentrades) {if(TradeIsOption && TradeIsLong) {var intrinsic = contractIntrinsic();if(intrinsic > 5) {
// At least $5 ITMcontractExercise(ThisTrade, 0);
}
}}

// Early exercise on dividendif(assetType(Asset) == 3) { // Stock// Check if there is a dividend
todayif(isDividendDate()) {for(opentrades) {if(TradeIsOption && TradeType ==
CALL)contractExercise(ThisTrade, 0);}}}
// Exercise if ITM
for(opentrades) {
if(TradeIsOption && TradeIsLong) {
var intrinsic = contractIntrinsic();
if(intrinsic > 5) { // At least $5 ITM
contractExercise(ThisTrade, 0);
```

```
}
}
}
// Early exercise on dividend
if(assetType(Asset) == 3) { // Stock
// Check if there is a dividend today
if(isDividendDate()) {
for(opentrades) {
if(TradeIsOption && TradeType == CALL)
contractExercise(ThisTrade, 0);
}
}
}
```

Note:\

- Not for European options before expiration\
- For ITM options only\
- In backtest: closes at intrinsic price\
- LIVE: Send exercise order to broker\
- Using contractCheck() to handle the resulting underlying

---

contractPosition - Current Contract Position

Description: Returns the net position (number of open contracts) of an option/future trade.

Syntax:

Note:

- Not for European options before expiration
- For ITM options only
- In backtest: closes at intrinsic price
- LIVE: Send exercise order to broker
- Use contractCheck() to handle the resulting underlying

2.8.4. contractPosition - Current Contract Position

Description:

Returns the net position (number of open contracts) of an option/futures trade.

Syntax:

var contractPosition(TRADE* tr);
var contractPosition(); // Current trade
var contractPosition(TRADE* tr);
var contractPosition(); // Current trade
Parameters:\

- tr (TRADE*): [Optional] Trade to check (NULL = current)

  Return Value: var - Number of contracts (negative for short, positive for long)

  Example:

Parameters:
- **tr** (TRADE*): [Optional] Trade to check (NULL = current)

Return Value:
var - Number of contracts (negative for short, positive for long)

Example:

```
// Check if contract is still open
TRADE* myTrade = enterShort();// ... after a while ...if(contractPosition(myTrade) != 0) {
printf("\nStill open: %.0f contracts",
contractPosition(myTrade));
}

// Loop on open tradesfor(opentrades) {var pos = contractPosition();if(pos == 0) {printf("\nTrade %s was closed externally", TradeID);exitTrade(); // Remove from list}}
// Check if the contract is still open
TRADE* myTrade = enterShort();
// ... after a while ...
if(contractPosition(myTrade) != 0) {
printf("\nStill open: %.0f contracts",
contractPosition(myTrade));
}
// Loop on open trades
for(opentrades) {
var pos = contractPosition();
if(pos == 0) {
printf("\nTrade %s was closed externally", TradeID);
exitTrade(); // Remove from list
}
}
```

Note:\

- Requires GETPOSITION from the broker\

- LIVE: real position on broker\
- In backtest: from simulation\
- 0 = contract closed/exercised

---

contractPrice - Contract Price Update

Description: Updates the bid/ask and underlying prices of a specific contract from the broker.

Syntax:

Note:

- Requires GETPOSITION from the broker
- LIVE: Real position on broker
- In backtest: from simulation
- 0 = contract closed/exercised

2.8.5. contractPrice - Contract Price Update

Description:

Update the bid/ask and underlying prices of a specific contract from the broker.

Syntax:

var contractPrice(TRADE* tr);
var contractPrice(); // Current contract

var contractPrice(TRADE* tr);

var contractPrice(); // Current contract

Parameters:\

- tr (TRADE*): [Optional] Trade to update

Return Value:var - Current bid price

Example:

Parameters:

- **tr** (TRADE*): [Optional] Trade to update

Return Value:

var - Current bid price

Example:

// Update before making decisions
contract(CALL, 4500, 30);var currentBid = contractPrice();printf("\nCurrent bid: %.2f", currentBid);

// Update all tradesfor(opentrades) {if(TradeIsOption) {contractPrice(ThisTrade);printf("\n%s: Bid %.2f Ask %.2f Und %.2f",TradeID,ContractBid,ContractAsk,ContractUnderlying);}}

// Decision based on updated price
if(contractPrice() < 0.50) { // Bid < $0.50exitTrade(); // Close for cheap}

// Update before making decisions

contract(CALL, 4500, 30);

var currentBid = contractPrice();

printf("\nCurrent bid: %.2f", currentBid);

// Update all trades

for(opentrades) {

if(TradeIsOption) {

contractPrice(ThisTrade);

printf("\n%s: Bid %.2f Ask %.2f Und %.2f",

TradeID,

ContractBid,

ContractAsk,

ContractUnderlying);

}

}

// Decision based on updated price

if(contractPrice() < 0.50) { // Bid < $0.50

exitTrade(); // Close for cheap

}

Note:\

- Update ContractBid, ContractAsk, ContractUnderlying, ContractVol\
- LIVE: Data from the broker\
- In backtest: from historical data\
- Open trade prices updated automatically

  ---

  contractRecord - Save Option Chain

  Description: Saves the current option chain to a .t8 file for future use in backtesting.

  Syntax:

Note:

- Update ContractBid, ContractAsk, ContractUnderlying, ContractVol
- LIVE: Data from the broker

- In backtest: from historical data
- Open trade prices updated automatically

2.8.6. contractRecord - Option Chain saving

Description:

Save the current option chain to a .t8 file for future use in backtesting.

Syntax:

int contractRecord();

int contractRecord();

Return Value:
int - Number of contracts saved

Example:

Return Value:

int - Number of saved contracts

Example:

// Save chain daily
function run() {if(is(LIVE) && hour() == 16) { // After closingasset("SPX");if(contractUpdate(Asset, 0, CALL|PUT)) {int saved = contractRecord();printf("\nSaved %d contracts", saved);}}}

// Use saved chain in backtestfunction run() {History = ".t8"; // Use option chain dataasset("SPX");

if(contractUpdate(Asset, 0, CALL|PUT)) {// Trading with historical data optionscontract(CALL, 4500, 30);enterShort();}}

```
// Save chain daily
function run() {
if(is(LIVE) && hour() == 16) { // After closing
asset("SPX");
if(contractUpdate(Asset, 0, CALL|PUT)) {
int saved = contractRecord();
printf("\nSaved %d contracts", saved);
}
}
}
// Use saved chain in backtest
function run() {
History = ".t8"; // Use option chain data
asset("SPX");
if(contractUpdate(Asset, 0, CALL|PUT)) {
```

// Trading with historical options data
contract(CALL, 4500, 30);
enterShort();
}
}
Note:\

- Save to History/Asset.t8\
- Includes prices, greeks, underlying\
- .t8 files can be used with History = ".t8"\
- Useful for precise backtesting on options

---

contractCPD - Price Probability Distribution

Description: Analyzes the probability distribution of the future price based on option prices (implied distribution).

Syntax:

Note:

- Save to History/Asset.t8
- Includes prices, greeks, underlying
- .t8 files can be used with History = ".t8"
- Useful for precise backtesting on options

2.8.7. contractCPD - Price Probability Distribution

Description:

Analyze the probability distribution of the future price based on option prices (implied distribution).

Syntax:

bool contractCPD(var Days, var MinPrice, var MaxPrice, int Steps);
bool contractCPD(var Days, var MinPrice, var MaxPrice, int Steps);
Parameters:\

- Days (var): Time horizon in days\
- MinPrice (var): Minimum price of the distribution.
- MaxPrice (var): Maximum price of the distribution\
- Steps (int): Number of steps in the distribution

Return Value: bool - TRUE if analysis successful

Example:

Parameters:

- **Days** (var): Time horizon in days
- **MinPrice** (var): Minimum price of the distribution
- **MaxPrice** (var): Maximum price of the distribution
- **Steps** (int): Number of steps in the distribution

Return Value:

bool - TRUE if analysis successful

Example:

```
// Analyze probability for the next 30 days
function run() {asset("SPX");contractUpdate(Asset, 0, CALL|PUT);

var price = priceClose();var range = 0.20 * price; // +/- 20%

if(contractCPD(30, price - range, price + range, 100)) {// Probability that price < current pricevar
probDown = cpd(price);printf("\nProb down: %.1f%%", probDown * 100);

// Price at the 25th percentilevar p25 = cpdv(0.25);printf("\n25th percentile: %.2f", p25);
}
}
// Analyze probabilities for the next 30 days
function run() {
asset("SPX");
contractUpdate(Asset, 0, CALL|PUT);
var price = priceClose();
var range = 0.20 * price; // +/- 20%
if(contractCPD(30, price - range, price + range, 100)) {
// Probability that price < current price
var probDown = cpd(price);
printf("\nProb down: %.1f%%", probDown * 100);
// Price at the 25th percentile
var p25 = cpdv(0.25);
printf("\n25th percentile: %.2f", p25);
}
}
```

Note:\

- Based on option implied volatility\
- More accurate near expiration\

- Requires liquid chain with many strikes

---

comboAdd - Add Contract to Combo

Description: Adds a contract to an existing multi-leg combo or starts a new one.

Syntax:

Note:

- Based on option implied volatility
- More accurate near expiration
- Requires liquid chain with many strikes

2.8.8. comboAdd - Add Contract to Combo

Description:

Adds a contract to an existing multi-leg combo or starts a new one.

Syntax:

void comboAdd(CONTRACT* c, int Lots);

void comboAdd(CONTRACT* c, int Lots);

Parameters:\

- c (CONTRACT*): Contract to add\
- Lots (int): Number of contracts (negative for short)

Return Value: void

Example:

Parameters:

- **c** (CONTRACT*): Contract to be added

- **Lots** (int): Number of contracts (negative for short)

Return Value:

void

Example:

```
// Iron Condor manual
function run() {contractUpdate("SPY", 0, CALL|PUT);var price = priceClose();

// Clear previous combocomboAdd(NULL, 0);

// Add the 4 legcomboAdd(contractFind(CALL, 45, price + 10), -1); //
SellcomboAdd(contractFind(CALL, 45, price + 20), 1); // BuycomboAdd(contractFind(PUT, 45, price -
```

10), -1); // SellcomboAdd(contractFind(PUT, 45, price - 20), 1); // Buy

// Open comboif(comboLegs() == 4) {MarginCost = comboMargin(-1, 3);for(int i = 1; i <= 4; i++) {if(i <= 2)enterShort(comboLeg(i));elseenterLong(comboLeg(i));}}}

```
// Iron Condor manual
function run() {
contractUpdate("SPY", 0, CALL|PUT);
var price = priceClose();
// Clear previous combo
comboAdd(NULL, 0);
// Add the 4 legs
comboAdd(contractFind(CALL, 45, price + 10), -1); // Sell
comboAdd(contractFind(CALL, 45, price + 20), 1); // Buy
comboAdd(contractFind(PUT, 45, price - 10), -1); // Sell
comboAdd(contractFind(PUT, 45, price - 20), 1); // Buy
// Open combo
if(comboLegs() == 4) {
MarginCost = comboMargin(-1, 3);
for(int i = 1; i <= 4; i++) {
if(i <= 2)
enterShort(comboLeg(i));
else
enterLong(comboLeg(i));
}
}
}
```

Note:\

- comboAdd(NULL, 0) clears current combo\
- Lots negative = short, positive = long\
- Use with enterLong/Short() on each leg

---

comboLegs - Leg Number in Combo

Description:Returns the leg number in the current combo.

Syntax:

Note:

- comboAdd(NULL, 0) clears current combo

- Lots negative = short, positive = long
- Use with enterLong/Short() on each leg

2.8.9. comboLegs - Leg Number in the Combo

Description:

Returns the number of legs in the current combo.

Syntax:

int comboLegs();

int comboLegs();

Return Value:
int - Number of legs

Example:

Return Value:

int - Number of legs

Example:

// Check combo complete
if(combo(...)) {if(comboLegs() != 4) {printf("\nIncomplete combo");return;}}

// Combo check complete

if(combo(...)) {

if(comboLegs() != 4) {

printf("\nIncomplete combo");

return;

}

}

MULTI-LEG STRATEGIES

combo - Multi-Leg Strategy Creation

Description: Create a multi-leg strategy (spread, straddle, iron condor, butterfly, etc.) in a single atomic operation.

Syntax:

2.9. MULTI-LEG STRATEGIES

2.9.1. combo - Multi-Leg Strategy Creation

Description:

Create a multi-leg strategy (spread, straddle, iron condor, butterfly, etc.) in a single atomic operation.

Syntax:

bool combo(CONTRACT* c1, int Lots1,
CONTRACT* c2, int Lots2,CONTRACT* c3, int Lots3,CONTRACT* c4, int Lots4);

bool combo(CONTRACT* c1, int Lots1,

CONTRACT* c2, int Lots2,

CONTRACT* c3, int Lots3,

CONTRACT* c4, int Lots4);

Parameters:\

- c1-c4 (CONTRACT*): Strategy Contracts\
- Lots1-4 (int): Number of contracts per leg (negative = short)

  Return Value: bool - TRUE if valid combo

  Example:

Parameters:

- **c1-c4** (CONTRACT*): Strategy contracts

- **Lots1-4** (int): Number of contracts per leg (negative = short)

Return Value:

bool - TRUE if combo valid

Example:

```
// Bull Put Spread
function run() {Hedge = 2;asset("SPY");contractUpdate("SPY", 0, PUT);

var price = priceClose();var expiry = contractExpiry(30);

// Sell PUT ATM, Buy PUT OTMif(combo(contractFind(PUT, 30, price), -1,contractFind(PUT, 30, price - 10), 1,0, 0, 0, 0)) {

MarginCost = comboMargin(-1, 3);
enterShort(comboLeg(1)); // Sell higher strike
enterLong(comboLeg(2)); // Buy lower strike}}

// Iron Condorfunction run() {var price = priceClose();var exp = contractExpiry(45);

if(combo(contractFind(CALL, 45, price + 50), -1,contractFind(CALL, 45, price + 100),
1,contractFind(PUT, 45, price - 50), -1,contractFind(PUT, 45, price - 100), 1)) {

MarginCost = comboMargin(-1,
```

```
3);enterShort(comboLeg(1));enterLong(comboLeg(2));enterShort(comboLeg(3));enterLong(comboLeg(4));}}
// Bull Put Spread
function run() {
Hedge = 2;
asset("SPY");
contractUpdate("SPY", 0, PUT);
var price = priceClose();
var expiry = contractExpiry(30);
// Sell PUT ATM, Buy PUT OTM
if(combo(
contractFind(PUT, 30, price), -1,
contractFind(PUT, 30, price - 10), 1,
0, 0, 0, 0)) {
MarginCost = comboMargin(-1, 3);
enterShort(comboLeg(1)); // Sell higher strike
enterLong(comboLeg(2)); // Buy lower strike
}
}
// Iron Condor
function run() {
var price = priceClose();
var exp = contractExpiry(45);
if(combo(
contractFind(CALL, 45, price + 50), -1,
contractFind(CALL, 45, price + 100), 1,
contractFind(PUT, 45, price - 50), -1,
contractFind(PUT, 45, price - 100), 1)) {
MarginCost = comboMargin(-1, 3);
enterShort(comboLeg(1));
enterLong(comboLeg(2));
enterShort(comboLeg(3));
enterLong(comboLeg(4));
}
}
```

Note:\

- Atomic execution (all or nothing)\
- Margin automatically calculated with comboMargin()\

- Use enterShort() to sell, enterLong() to buy\
- exitCombo() closes the entire strategy

---

comboLeg - Combo Leg Selection

Description: Selects a specific leg of the combo to operate on.

Syntax:

Note:

- Atomic execution (all or nothing)
- Margin automatically calculated with comboMargin()
- Use enterShort() to sell, enterLong() to buy
- exitCombo() closes the entire strategy

2.9.2. comboLeg - Combo Leg Selection

Description:

Select a specific leg of the combo to operate on.

Syntax:

int comboLeg(int LegNum);
int comboLeg(var Strike, int Type, var Expiry); // Alternative

int comboLeg(int LegNum);

int comboLeg(var Strike, int Type, var Expiry); // Alternative

Parameters:\

- LegNum (int): Leg number (1-4)\
- Strike/Type/Expiry: Alternative to identify leg

Return Value: int - Number of selected leg, 0 if not found

Example:

Parameters:

- **LegNum** (int): Leg number (1-4)

- **Strike/Type/Expiry** : Alternative to identify leg

Return Value:

int - Number of selected leg, 0 if not found

Example:

// Use LegNum
enterShort(comboLeg(1)); // Operate on first legenterLong(comboLeg(2)); // Operate on second leg

// Use parametersvar price = priceClose();var exp = contractExpiry(30);

enterShort(comboLeg(price, CALL, exp));enterShort(comboLeg(price, PUT, exp));
// Use LegNum
enterShort(comboLeg(1)); // Operate on first leg
enterLong(comboLeg(2)); // Operates on second leg
// Use parameters
var price = priceClose();
var exp = contractExpiry(30);
enterShort(comboLeg(price, CALL, exp));
enterShort(comboLeg(price, PUT, exp));

ADVANCED TRADING

enterTrade - Insert Trade from Template

Description:
Inserts a trade from an existing TRADE structure. Used to synchronize with brokers or recreate positions.

Syntax:

2.10. ADVANCED TRADING

2.10.1. enterTrade - Insert Trade from Template

Description:

Inserts a trade from an existing TRADE structure. Used to synchronize with brokers or recreate positions.

Syntax:

TRADE* enterTrade(TRADE* Template);

TRADE* enterTrade(TRADE* Template);

Parameters:\

- Template (TRADE*): TRADE structure with parameters

  Return Value: TRADE* - Pointer to the new trade

  Example:

Parameters:

- **Template** (TRADE*): TRADE structure with parameters

Return Value:

TRADE* - Pointer to new trade

Example:

```
// Sync with broker
function main() {brokerTrades(0); // Load positions from broker

// Positions are now in trades Zorrofor(opentrades) {printf("\nSynced: %s %.0f @ %.2f",Asset,
TradeUnits, TradePriceOpen);}}

// Recreate position manually TRADE tr;
tr.flags = TRLONG;
tr.fUnits = 100;tr.fEntryPrice = 150.50;// ... other fields ...

enterTrade(&tr); // Insert into trade list
// Synchronize with broker
function main() {
brokerTrades(0); // Load positions from broker
// Positions are now in Zorro trades
for(opentrades) {
printf("\nSynced: %s %.0f @ %.2f",
Assets, TradeUnits, TradePriceOpen);
}
}
// Recreate position manually
TRADE tr;
tr.flags = TRLONG;
tr.fUnits = 100;
tr.fEntryPrice = 150.50;
// ... other fields ...
enterTrade(&tr); // Inserts trades into the list
```

Note:\

- Does not send order to broker\
- Internal insertion only\
- Mainly used by brokerTrades()\
- Useful for inherited or external positions

---

cancelTrade - Cancel Trade

Description: Cancels a pending trade or removes a trade from the list without closing it at the broker.

Syntax:

Note:

- Does not send order to broker
- Internal insertion only
- Mainly used by brokerTrades()
- Useful for inherited or external positions

2.10.2. cancelTrade - Trade Cancellation

Description:

Cancel a pending trade or remove a trade from the list without closing it at the broker.

Syntax:

void cancelTrade(TRADE* tr);

void cancelTrade(TRADE* tr);

Parameters:\

- tr(TRADE*): Trade to delete (NULL = current in loop)

Return Value: void

Example:

Parameters:

- **tr** (TRADE*): Trade to delete (NULL = current in the loop)

Return Value:

void

Example:

```
// Clear old pending orders
for(opentrades) {if(TradeIsPending &&TradeOpenBar - Bar > 5) { // Pending for 5+ bars
cancelTrade();}}

// Remove manually closed trade on broker TRADE* myTrade = enterLong(); // ... manually closed trade on broker ... cancelTrade(myTrade); // Remove from Zorro list
// Delete old pending orders
for(opentrades) {
if(TradeIsPending &&
TradeOpenBar - Bar > 5) { // Pending from 5+ bars
cancelTrade();
}
```

}

// Remove closed trade manually on broker

TRADE* myTrade = enterLong();

// ...closed manually on the broker...

cancelTrade(myTrade); // Remove Zorro from list

Note:\

- For pending orders: send DOCANCEL to the broker\
- For open trades: removal from list only\
- NULL clears the current trade in the loop\
- Does not close actual position

---

findTrade - Search Trade by Criteria

Description: Finds a trade in the open list that matches specific criteria.

Syntax:

Note:

- For pending orders: send DOCANCEL to the broker
- For open trades: removal from list only
- NULL clears the current trade in the loop
- Does not close actual position

2.10.3. findTrade - Trade Search by Criteria

Description:

Find a trade in the open list that matches specific criteria.

Syntax:

TRADE* findTrade(string AssetName, int AlgoID);

TRADE* findTrade(string AssetName, int AlgoID);

Parameters:\

- AssetName (string): Asset name (NULL = any)\
- AlgoID (int): Algorithm ID (0 = any)

Return Value: TRADE* - First trade found, NULL if none

Example:

Parameters:

- **AssetName** (string): Asset name (NULL = any)

- **AlgoID** (int): algorithm ID (0 = any)

Return Value:

TRADE* - First trade found, NULL if none

Example:

```
// Find trade on specific asset
TRADE* eurTrade = findTrade("EUR/USD", 0);if(eurTrade) {printf("\nFound EUR/USD trade: %.2f", TradeProfit);}

// Find by algoalgo("Strategy1");TRADE* t = findTrade(NULL, Algo);if(t) exitTrade(t);

// Close all trades of an assetwhile(findTrade("GBP/USD", 0))exitTrade(findTrade("GBP/USD", 0));
// Find trades on a specific asset
TRADE* eurTrade = findTrade("EUR/USD", 0);
if(eurTrade) {
printf("\nFound EUR/USD trade: %.2f", TradeProfit);
}
// Find by algo
algo("Strategy1");
TRADE* t = findTrade(NULL, Algo);
if(t) exitTrade(t);
// Close all trades of an asset
while(findTrade("GBP/USD", 0))
exitTrade(findTrade("GBP/USD", 0));
```

Note:\

- NULL for AssetName = search for any asset\
- AlgoID = 0 matches any algo\
- Return first found\
- Use in loop to find all

---

tradeUpdate - Update Trade Parameters

Description:
Changes the parameters of an open trade (lots, stops, targets, etc.).

Syntax:

Note:

- NULL for AssetName = search for any asset
- AlgoID = 0 matches any algo

- Return first found
- Use in loop to find all

2.10.4. tradeUpdate - Trade Parameters Update

Description:

Change the parameters of an open trade (lots, stops, targets, etc.).

Syntax:

bool tradeUpdate(TRADE* tr, var NewLots, var NewStop, var NewProfit);

bool tradeUpdate(TRADE* tr, var NewLots, var NewStop, var NewProfit);

Parameters:\

- tr (TRADE*): Trade to update (NULL = current)\
- NewLots(var): New size (0 = do not change)\
- NewStop (var): New stop in pips (0 = do not change)\
- NewProfit(var): New target in pips (0 = do not change)

  Return Value: bool - TRUE if update successful

  Example:

Parameters:

- **tr** (TRADE*): Trade to update (NULL = current)

- **NewLots** (var): New size (0 = do not change)

- **NewStop** (var): New stop in pips (0 = do not change)

- **NewProfit** (var): New target in pips (0 = do not change)

Return Value:

bool - TRUE if update successful

Example:

```
// Manual trailing stop
for(opentrades) {if(TradeIsLong && TradeProfit > 0) {var newStop = priceClose(0) - 20*PIP; // 20 pips
trailif(newStop > TradePriceOpen + TradeStopLimit) {// Move stop only if
tradeUpdate(ThisTrade, 0,
(newStop - TradePriceOpen)/PIP, 0);}}}

// Scale in/outTRADE* myTrade = enterLong(10);// ... after profit ...tradeUpdate(myTrade, 5, 0, 0); //
Halve position

// Break-even stopfor(opentrades) {if(TradeProfit > 50*PIP) {tradeUpdate(ThisTrade, 0, 0.5*PIP, 0);
// Stop BE}}
// Manual trailing stop
for(opentrades) {
```

```
if(TradeIsLong && TradeProfit > 0) {
var newStop = priceClose(0) - 20*PIP; // 20 pips trail
if(newStop > TradePriceOpen + TradeStopLimit) {
// Move stop only if it improves
tradeUpdate(ThisTrade, 0,
(newStop - TradePriceOpen)/PIP, 0);
}
}
}
// Scale in/out
TRADE* myTrade = enterLong(10);
// ... after profit ...
tradeUpdate(myTrade, 5, 0, 0); // Halve position
// Break-even stop
for(opentrades) {
if(TradeProfit > 50*PIP) {
tradeUpdate(ThisTrade, 0, 0.5*PIP, 0); // Stop BE
}
}
```

Note:\

- 0 = do not change that parameter\
- Stop in pips from entry price\
- Some brokers do not support all exchanges\
- Check with specific broker

---

brokerTrades - Broker Position Synchronization

Description: Loads open positions from the broker and synchronizes them with the Zorro trade list.

Syntax:

Note:

- 0 = do not change that parameter
- Stop in pips from entry price
- Some brokers do not support all exchanges
- Check with specific broker

2.10.5. brokerTrades - Broker Position Synchronization

Description:

Upload open positions from the broker and synchronize them with the Zorro trade list.

Syntax:

int brokerTrades(int Filter);

int brokerTrades(int Filter);

Parameters:\

- Filter (int): Loading filter\
- 0 = all positions\
- 1 = only assets in asset list\
- 2 = only assets used in the script\
- 4 = current asset only (does not delete existing trades)\
- 16 = adjust for Multiplier (options)

  Return Value:int - Number of positions loaded

  Example:

Parameters:

- **Filter** (int): Loading filter

- 0 = all positions
- 1 = only assets in asset list
- 2 = only assets used in the script
- 4 = Current asset only (does not delete existing trades)
- 16 = adjust for Multiplier (options)

Return Value:

int - Number of positions loaded

Example:

```
// Synchronize on startup
function main() {int loaded = brokerTrades(0);printf("\nLoaded %d positions from broker", loaded);

// Trades are now visible for(opentrades) {printf("\n%s: %.0f @ %.2f",Asset, TradeUnits,
TradePriceOpen);}}

// List assets onlyassetList("MyAssets");brokerTrades(1); // Only load assets into MyAssets.csv

// Update positions without deleting trades ZorrobrokerTrades(4); // Update only, no cancel
// Sync on startup
function main() {
int loaded = brokerTrades(0);
printf("\nLoaded %d positions from broker", loaded);
```

```
// Trades are now visible
for(opentrades) {
printf("\n%s: %.0f @ %.2f",
Assets, TradeUnits, TradePriceOpen);
}
}
// Only assets in the list
assetList("MyAssets");
brokerTrades(1); // Only load assets into MyAssets.csv
// Update positions without deleting Zorro trades
brokerTrades(4); // Update only, no cancel
```

Note:\

- Requires GETTRADES from broker\
- Delete existing trades (except with Filter = 4)\
- Uploaded trades are "plain" (no TMF, flags, etc.)\
- Algo ID set to Current Algo

BROKER INTERACTION

brokerAccount - Broker Account Status

Description: Retrieves broker account information: balance, equity, margin used, etc.

Syntax:

Note:

- Requires GETTRADES from the broker
- Delete existing trades (except with Filter = 4)
- Uploaded trades are "plain" (no TMF, flags, etc.)
- Algo ID set to Current Algo

2.11. BROKER INTERACTION

2.11.1. brokerAccount - Broker Account Status

Description:

Retrieve broker account information: balance, equity, margin used, etc.

Syntax:

```
var brokerAccount(string Name, int ID, var* pEquity, var* pMargin);
```

var brokerAccount(string Name, int ID, var* pEquity, var* pMargin);

Parameters:\

- Name (string): Account name (NULL = current)\
- ID (int): [Optional] Account ID\
- pEquity (var*): [Output] Current Equity\
- pMargin(var*): [Output] Margin used

  Return Value:var - Account Balance

  Example:

Parameters:

- **Name** (string): Account name (NULL = current)

- **ID** (int): [Optional] Account ID

- **pEquity** (var*): [Output] Current Equity

- **pMargin** (var*): [Output] Margin used

Return Value:

var - Account Balance

Example:

```
// Get current account info
var equity, margin;var balance = brokerAccount(NULL, 0, &equity, &margin);

printf("\nBalance: %.2f", balance);printf("\nEquity: %.2f", equity);printf("\nMargin: %.2f", margin);printf("\nFree: %.2f", equity - margin);

// Multi-broker setupvar bal1 = brokerAccount("IB", 0, NULL, NULL);var bal2 = brokerAccount("Oanda", 0, NULL, NULL);

printf("\nTotal balance: %.2f", bal1 + bal2);
// Retrieve current account info
var equity, margin;
var balance = brokerAccount(NULL, 0, &equity, &margin);
printf("\nBalance: %.2f", balance);
printf("\nEquity: %.2f", equity);
printf("\nMargin: %.2f", margin);
printf("\nFree: %.2f", equity - margin);
// Multi-broker setup
var bal1 = brokerAccount("IB", 0, NULL, NULL);
var bal2 = brokerAccount("Oanda", 0, NULL, NULL);
printf("\nTotal balance: %.2f", bal1 + bal2);
```

Note:\

- Automatically called by Zorro\
- Optional for broker plugin\
- If not implemented: Zorro estimates from trade\
- NULL for unused pointers

---

brokerAsset - Asset Parameters from Broker

Description:
Downloads an asset's trading parameters directly from the broker (spread, volume, availability).

Syntax:

Note:

- Automatically called by Zorro
- Optional for broker plugin
- If not implemented: Zorro trade estimate
- NULL for unused pointers

- **brokerAsset - Asset parameters from the Broker**

Description:

Download an asset's trading parameters directly from the broker (spread, volume, availability).

Syntax:

int brokerAsset(string Symbol, var* pPrice, var* pSpread, var* pVolume);

int brokerAsset(string Symbol, var* pPrice, var* pSpread, var* pVolume);

Parameters:\

- Symbol (string): Broker symbol of the asset\
- pPrice (var*): [Output] Current price\
- pSpread (var*): [Output] Current spread\
- pVolume (var*): [Output] Volume or availability

Return Value:int - 1 if available, 0 if not found

Example:

Parameters:

- **Symbol** (string): Broker symbol of the asset
- **pPrice** (var*): [Output] Current price
- **pSpread** (var*): [Output] Current spread
- **pVolume** (var*): [Output] Volume or availability

Return Value:

int - 1 if available, 0 if not found

Example:

```
// Check asset availability
var price, spread, volume;if(brokerAsset("EUR/USD", &price, &spread, &volume))
{printf("\nEUR/USD available");
printf("\nPrice: %.5f", price);
printf("\nSpread: %.5f", spread);} else {printf("\nEUR/USD not available for trading");}

// Update spread dynamicfunction run() {var spread;if(brokerAsset(SymbolLive, NULL, &spread,
NULL)) {Spread = spread; // Use live spreads}}
// Check asset availability
var price, spread, volume;
if(brokerAsset("EUR/USD", &price, &spread, &volume)) {
printf("\nEUR/USD available");
printf("\nPrice: %.5f", price);
printf("\nSpread: %.5f", spread);
} else {
printf("\nEUR/USD not available for trading");
}
// Dynamic spread update
function run() {
var spread;
if(brokerAsset(SymbolLive, NULL, &spread, NULL)) {
Spread = spread; // Use live spreads
}
}
```

Note:\

- Automatically called for selected assets\
- Spread in price units (not pips)\
- Volume may be residual availability\
- NULL for unrequired parameters

---

brokerCommand - Special Broker Commands

Description: Sends special commands to the broker plugin for advanced features not covered by the standard functions.

Syntax:

Note:

- Automatically called for selected assets
- Spread in price units (not pips)
- Volume may be residual availability
- NULL for unrequired parameters

- **brokerCommand - Special Broker Commands**

Description:

Send special commands to the plugin broker for advanced features not covered by the standard functions.

Syntax:

var brokerCommand(int Command, DWORD Parameter);
long brokerCommand(int Command, DWORD Parameter); // Long version

var brokerCommand(int Command, DWORD Parameter);

long brokerCommand(int Command, DWORD Parameter); // Long version

Parameters:\

- Command (int): Command code (see list below)\
- Parameter (DWORD): Command-dependent parameter

Main Commands:\

- GETACCOUNT = 0: Account Balance\
- GETPOSITION = 1: Net asset position\
- SETSLIPPAGE = 2: Maximum slippage\
- GETMARGINAVAIL = 3: Margin available\
- GETSPREAD = 4: Current spread\
- GETCOMPLIANCE = 5: Broker connected?
- GETNTRADES = 6: Number of open trades\
- SETORDERTYPE = 7: Order type (FOK, IOC, GTC)\
- SETPRICETYPE = 8: Price type (BID, ASK, TRADE)\
- SETMAGIC = 9: Magic number for identification\
- GETTRADES = 10: Load positions (see brokerTrades)\
- DOCANCEL = 11: Cancel order\
- GETPOSITION = 12: Check specific position

Return Value: var/long - Depends on command

Example:

Parameters:

- **Command** (int): Command code (see list below)

- **Parameter** (DWORD): Command-dependent parameter

Main Commands:

- GETACCOUNT = 0: Account balance
- GETPOSITION = 1: Net asset position
- SETSLIPPAGE = 2: Maximum slippage
- GETMARGINAVAIL = 3: Margin available
- GETSPREAD = 4: Current spread
- GETCOMPLIANCE = 5: Broker connected?
- GETNTRADES = 6: Number of open trades
- SETORDERTYPE = 7: Order type (FOK, IOC, GTC)
- SETPRICETYPE = 8: Price type (BID, ASK, TRADE)
- SETMAGIC = 9: Magic number for identification
- GETTRADES = 10: Load positions (see brokerTrades)
- DOCANCEL = 11: Cancel order
- GETPOSITION = 12: Check specific position

Return Value:

var/long - Depends on the command

Example:

```
// Check connection
if(!brokerCommand(GETCOMPLIANCE, 0)) {printf("\nBroker not connected!");return;}

// Get balancevar balance = brokerCommand(GETACCOUNT, 0);printf("\nBalance: $%.2f", balance);

// Net position on current assetvar position = brokerCommand(GETPOSITION, 0);printf("\nNet position: %.0f lots", position);

// Set slippage massimobrokerCommand(SETSLIPPAGE, 5); // Max 5 pips

// Available marginvar freeMargin = brokerCommand(GETMARGINAVAIL, 0);

// Set order type
brokerCommand(SETORDERTYPE, 2); // GTC orders

// Magic number for tradebroker identificationCommand(SETMAGIC, 12345);
// Check connection
if(!brokerCommand(GETCOMPLIANCE, 0)) {
printf("\nBroker not connected!");
return;
```

```
}
// Get balance
var balance = brokerCommand(GETACCOUNT, 0);
printf("\nBalance: $%.2f", balance);
// Net position on current asset
var position = brokerCommand(GETPOSITION, 0);
printf("\nNet position: %.0f lots", position);
// Set maximum slippage
brokerCommand(SETSLIPPAGE, 5); // Max 5 pips
// Available margin
var freeMargin = brokerCommand(GETMARGINAVAIL, 0);
// Set order type
brokerCommand(SETORDERTYPE, 2); // GTC orders
// Magic number for trade identification
brokerCommand(SETMAGIC, 12345);
```

Note:\

- Available commands depend on the broker\
- Some only work in LIVE mode\
- Consult specific broker documentation\
- Use with caution

Notes:

- Available commands depend on the broker
- Some only work in LIVE mode
- Consult specific broker documentation
- Use with caution

3. CHAPTER 3: MATH, TIME & DATA

Mathematical and utility functions form the basis for advanced calculations in Zorro. This chapter presents comprehensive documentation of all the available functions for mathematical operations, statistics, fuzzy logic, and conversions.

MAIN CATEGORIES:

- 📊 Basic Math: abs, ceil, cos, exp, floor, log, pow, round, sin, sqrt, tan
- 📈 Advanced Math: between, clamp, ifelse, erf, modf, dnorm, roundto
- 📉 Statistics: cdf, qnorm, random, andF
- 🔀 Fuzzy Logic: fuzzy, orF
- 📅 Time & Date: at, day, dmy, dow, hour, minute, month, tod, wdate, year, ymd

- ⏰ Advanced Date/Time: ldate, ldow, lhour, ltod, ltow, ndow, workday, week, sun, tdm, tom, dst
- 📝 Strings: strcat, strcmp, strcpy
- 🔤 Advanced Strings: strcon, strtr, stridx, strxid, strw, sftoa, strxc

3.1. 📊 BASIC MATHEMATICS

■ **abs** - Absolute Value

Returns the absolute value of a number. Essential for distance calculations, spreads, and risk management.

var abs(var Value);

→ var - Absolute value always >= 0

Example: var distance = abs(Price[0] - MA);

Note: Essential for directionless comparisons • Useful for floating-point tolerances

■ **ceil** - Rounding Up

Returns the smallest integer >= value. ceil(4.2) = 5, ceil(-4.7) = -4.

int ceil(var Value);

→ int - Smallest integer >= Value

Example: int Lots = ceil(2.3); // = 3

Note: Ensures minimum values • For negative values approach zero

■ **cos** - Cosine

Calculates the cosine of an angle in radians. It ranges from -1 to +1. Essential for cycle analysis and pattern matching.

var cos(var Angle);

→ var - Cosine between -1.0 and +1.0

Example: var CosWave = cos(Bar * 2 * PI / 24);

Note: Angles in radians not degrees • cos(0)=1, cos(PI/2)=0 • Even function: cos(-x)=cos(x)

■ **exp** - Exponential Function

Calculate e^x. Fundamentals in statistics, finance (compound interest), and machine learning.

var exp(var Exponent);

→ var - e raised to the exponent (always > 0)

Example: var Growth = exp(0.05 * Years);

Note: exp(0)=1, exp(1)≈2.718 • Opposite of log() • Beware of overflow for values > 700

■ **floor** - Round Down

Returns the largest integer <= value. floor(4.7) = 4, floor(-3.2) = -4.

int floor(var Value);

→ int - Largest integer <= Value

Example: int SafeLots = floor(2.8); // = 2

Note: Conservative approach • For negatives move away from zero

■ **log** - Natural Logarithm

Calculates the natural logarithm (base e). Inverse of exp(). Used for logarithmic returns and normalizations.

var log(var Value);

→ var - Natural logarithm (Value must be > 0)

Example: var LogReturn = log(Price[0] / Price[1]);

Note: Only for values > 0 • log(1)=0, log(e)=1 • Log returns are time-additive

■ **pow** - Power

Calculates base^exponent. Essential for compound growth calculations, normalizations, and roots.

var pow(var Base, var Exponent);

→ var - Base raised to the exponent

Example: var FinalValue = Principal * pow(1.05, 10);

Note: pow(x, 0.5) = sqrt(x) • pow(x, -1) = 1/x • For base=e, prefer exp()

■ **round** - Rounding

Round to the nearest whole number. 4.4→4, 4.5→5, 4.6→5. Standard rule of thumb: 0.5 rounds up.

int round(var Value);

→ int - Rounded value

Example: int Lots = round(2.6); // = 3

Note: round(4.5) = 5 • For decimal precision use roundto()

■ **sin** - Sinus

Calculates the sine of an angle in radians. It ranges from -1 to +1. Essential for cycle and pattern analysis.

var sin(var Angle);

→ var - Sine between -1.0 and +1.0

Example: var SineWave = sin(Bar * 2 * PI / 20);

Note: Angles in radians • sin(0)=0, sin(PI/2)=1 • Odd function: sin(-x)=-sin(x)

■ **sqrt** - Square Root

Calculates the square root. Essential for Euclidean distances, standard deviations, and volatility.

var sqrt(var Value);

→ var - Square root (Value must be >= 0)

Example: var StdDev = sqrt(SumSq / N);

Note: Only for values >= 0 • sqrt(x*x) = |x| • Volatility scaling: σt = σ1 * sqrt(t)

■ **tan** - Tangent

Calculate the tangent. tan(x) = sin(x)/cos(x). It has period π and asymptotes at ±π/2. Range: -∞ to +∞.

var tan(var Angle);

→ var - Tangent (unlimited range)

Example: var Slope = tan(TrendAngle);

Note: Has asymptotes where cos(x)=0 • tan(π/4)=1 • Odd function

3.2. 📈 ADVANCED MATHEMATICS

■ **between** - Range Control

Checks if value is in [Min, Max]. TRUE if Min <= Value <= Max.

bool between(var Value, var Min, var Max);

→ bool - TRUE if in range

Example: if(between(rsi, 40, 60)) { }

Note: Extremes included • More readable than (Val>=Min && Val<=Max)

■ **clamp** - Value Limitation

Limit value to [Min, Max]. If <Min returns Min, if >Max returns Max, otherwise Value.

var clamp(var Value, var Min, var Max);

→ var - Range-limited value

Example: var Size = clamp(Calculated, 1, 10);

Note: Prevents out-of-range values • Equivalent to max(Min, min(Max, Value))

■ **ifelse** - Conditional Assignment

Ternary operator. Returns TrueValue if Condition is TRUE, otherwise FalseValue.

var ifelse(bool Condition, var TrueValue, var FalseValue);

→ var - TrueValue or FalseValue

Example: var Size = ifelse(vol > 0.02, 1, 2);

Note: Lazy evaluation • More compact than if-else

■ **erf** - Error Function

Gaussian error function. erf(x) = (2/√π) $\int_0^x$ e^(-t²) dt. Varies from -1 to +1. Related to normal CDF.

var erf(var x);

→ var - Value between -1 and +1

Example: var Prob = 0.5 * (1 + erf(z/sqrt(2)));

Note: erf(0)=0 • cdf(x) = 0.5*(1+erf(x/√2))

■ **modf** - Module for Floating-Point

Divides a number into a whole number and a fractional number. Returns the fractional part and stores the whole number in the pointer.

var modf(var Value, var* IntegerPart);

→ var - Fractional part with the same sign as Value

Example: var frac = modf(123.45, &integer);

Note: Fractional part has same sign • modf(-3.14, &x) → x=-3.0, returns -0.14

■ **dnorm** - Normal Density

Probability density normal distribution with mean μ and deviation σ. Standard Gaussian formula.

var dnorm(var x, var Mean, var StdDev);

→ var - Probability density >= 0

Example: var dens = dnorm(Price, Mean, StdDev);

Note: StdDev must be > 0 • Integral over R is 1

■ **roundto** - Round to Precision

Round to a specific multiple. roundto(1.234, 0.1) = 1.2, roundto(123, 5) = 125.

var roundto(var Value, var Precision);

→ var - Value rounded to step Precision

Example: var priceRnd = roundto(Price, 0.25);

Notes: roundto(x, 1) = round(x) • Precision must be > 0

3.3. 📉 STATISTICS

■ **cdf** - Normal CDF

Standard normal cumulative distribution function. P(Z ≤ x). Varies from 0 to 1, cdf(0) = 0.5.

var cdf(var x);

→ var - Cumulative probability 0-1

Example: var prob = cdf((Price - Mean) / StdDev);

Note: cdf(1.96)≈0.975 • Inverse of qnorm()

■ **qnorm** - Normal Quantile

Quantile (z-value) for given probability. Inverse of cdf(). qnorm(0.975) ≈ 1.96.

var qnorm(var Probability);

→ var - corresponding Z-score

Example: var zcrit = qnorm(0.975); // 1.96

Note: Probability must be (0, 1) • qnorm(0.5)=0

■ **random** - Random Number

Generates a uniform pseudo-random number. Without parameter: [0,1). With Limit: [0, Limit).

var random();
var random(var Limit);

→ var - Uniform Random Number

Example: var r = random(); // 0 to 1

Note: Use seed() for reproducibility • For normal: qnorm(random())

3.4. 🔀 FUZZY LOGIC

■ **andF** - Fuzzy AND

Fuzzy AND operator: andF(A, B) = min(A, B). Fuzzy logic with values 0-1.

var andF(var A, var B);

→ var - Minimum between A and B

Example: var signal = andF(TrendUp, HighVol);

Note: T-norm standard • More conservative than OR

■ **fuzzy** - Defuzzification

Converts a fuzzy value (0-1) to a Boolean. TRUE if >= 0.5, FALSE if < 0.5.

bool fuzzy(var Value);

→ bool - TRUE/FALSE

Example: bool trigger = fuzzy(SignalStrength);

Note: Threshold at 0.5 • Simple defuzzification

■ **orF** - OR Fuzzy

Fuzzy OR operator: orF(A, B) = max(A, B). Fuzzy logic with values 0-1.

var orF(var A, var B);

→ var - Maximum between A and B

Example: var alert = orF(Overbought, Oversold);

Note: Standard T-conorm • Less restrictive than AND

3.5. TIME & DATE - Basic Functions

■ **at** - Time-Specific Event

Checks whether an event occurs at a specific time (HHMM format). TRUE only on the first bar of that time.

bool at(int Time);

→ bool - TRUE if it is the specified time

Example: if(at(930)) // 9:30 AM

Notes: Format HHMM (0-2359) • TRUE first occurrence only

■ **day** - Day of the Month

Returns the day of the month (1-31) for the current or offset bar.

int day(int Offset=0);

→ int - Day of the month 1-31

Example: int today = day(); // e.g. 15

Note: Offset: 0=current, 1=previous • Timezone UTC

■ **dmy** - Conversion YYYYMMDD → DATE

Converts date format YYYYMMDD (integer) to DATE (OLE time). Ex: 20250115 → DATE.

DATE dmy(int YYYYMMDD);

→ DATE - OLE time/date value

Example: DATE myDate = dmy(20250115);

Note: Format: year*10000 + month*100 + day • Inverse of ymd()

■ **dow** - Day of the Week

Returns day of the week: 1=Monday, 2=Tuesday, ..., 5=Friday, 6=Saturday, 7=Sunday.

int dow(int Offset=0);

→ int - Weekday 1-7

Example: if(dow() == 5) // Friday

Notes: 1=Mon, 5=Fri, 7=Sun • Timezone UTC

■ **hour** - UTC time

Returns the current time in UTC (0-23).

int hour();

→ int - Time 0-23

Example: if(hour() >= 9 && hour() < 17)

Note: UTC timezone • For local time use lhour()

■ **minute** - Minute

Returns the minutes of the current hour (0-59).

int minute();

→ int - Minutes 0-59

Example: if(minute() == 0) // every hour

Note: UTC timezone • Combine with hour() for precise time

■ **month** - Month

Returns the current month (1-12): 1=January, ..., 12=December.

int month(int Offset=0);

→ int - Month 1-12

Example: if(month() == 12) // December

Notes: Offset: 0=current • 1=Jan, 12=Dec

■ **tod** - Time of Day HHMM

Returns the time of day in HHMM (0-2359) format. Ex: 1530 = 15:30.

int tod();

→ int - Time format HHMM

Example: if(tod() == 1530) // 3:30 PM

Note: Format HHMM • UTC timezone

■ **wdate** - OLE Time/Date

Returns the current date/time in OLE format (double). Days since 12/30/1899, fraction = time.

DATE wdate();

→ DATE - OLE time/date

Example: DATE now = wdate();

Note: OLE format: days.fraction • Use for time calculations

■ **year** - Year

Returns the current year (e.g. 2025) with a possible offset.

int year(int Offset=0);

→ int - 4-digit year

Example: if(year() == 2025)

Note: Offset: 0=current • Full year (not 2-digit)

■ **ymd** - Conversion DATE → YYYYMMDD

Converts DATE (OLE time) to YYYYMMDD (integer) format. Inverse of dmy().

int ymd(DATE Date);

→ int - Date format YYYYMMDD

Example: int dateInt = ymd(wdate());

Note: Format: year*10000 + month*100 + day • Inverse of dmy()

3.6. ADVANCED DATE/TIME

■ **ldate** - Local Date

Returns the current date/time in the local timezone (not UTC).

DATE ldate();

→ DATE - local OLE time/date

Example: DATE localNow = ldate();

Note: Considers timezone and DST • Different from wdate()

■ **ldow** - Local Day of Week

Day of the week in local timezone (1-7).

int ldow(int Offset=0);

→ int - Local weekday 1-7

Example: if(ldow() == 5) // Local Ven

Notes: Local Timezone • 1=Mon, 7=Sun

■ **lhour** - Local Time

Current time in local timezone (0-23).

int lhour(int Offset=0);

→ int - Local time 0-23

Example: if(lhour() == 9) // 9 AM local

Note: Consider DST • Different from hour()

■ **ltod** - Local Time of Day

Time of day format HHMM in local timezone.

int ltod(int Offset=0);

→ int - local HHMM

Example: if(ltod() == 930) // 9:30 local

Note: HHMM format • Local timezone

■ **ltow** - Local Time of Week

Local DHHMM weekday time. D=1-7, HH=00-23, MM=00-59.

int ltow(int Offset=0);

→ int - DHHMM (e.g. 10930 = Mon 09:30)

Example: if(ltow() >= 10900)

Notes: D: 1=Mon, 7=Sun • Local timezone

■ **ndow** - N-th Day of the Week

Returns the DATE of the specific nth day of the month. Ex: 3rd Friday of the month.

DATE ndow(int WeekDay);

→ DATE - Date of the nth weekday

Example: DATE thirdFri = ndow(35); // 3rd Fri

Notes: Format: N*10 + WeekDay • N=1-5, WeekDay=1-7

■ **workday** - Working Day

Checks whether date is a weekday (not a weekend/holiday). Uses the Holidays array.

bool workday(DATE Date);

→ bool - TRUE if workday

Example: if(workday(wdate()))

Note: Check weekend • Use Holidays array if set

■ **week** - Week Number

Returns week number of year (1-53) according to ISO 8601.

int week(int Offset=0);

→ int - Week 1-53

Example: int weekNum = week();

Note: ISO 8601 standard • Week starts on Monday

■ **dom** - Days in the Month

Returns number of days in the current month (28-31).

int dom(int Offset=0);

→ int - Days in month 28-31

Example: int daysInMonth = sun();

Note: Consider leap years • 28=Feb non-leap, 29=Feb leap

■ **tdm** - Trading Days in the Month

Number of trading days in the month (excludes weekends/holidays).

int tdm(int Offset=0);

→ int - Trading days ~20-23

Example: int tradeDays = tdm();

Note: Excludes weekends • Use Holidays if available

■ **tom** - Trading Days This Month

Trading days spent in the current month.

int tom();

→ int - Trading days elapsed

Example: if(tom() == 1) // first day of trading

Note: Increases every trading day • Resets at the beginning of the month

■ **dst** - Daylight Saving Time

Checks whether the date is in DST (Daylight Saving Time). TRUE if DST is on.

int dst(DATE Date);

→ int - 1 if DST, 0 otherwise

Example: if(dst(wdate()))

Note: Depends on timezone • Consider local DST rules

■ **minutesAgo** - Minutes from Time

Calculate minutes elapsed since a specific time HHMM today.

int minutesAgo(int HHMM);

→ int - Minutes elapsed

Example: int elapsed = minutesAgo(930);

Notes: HHMM format • Negative if future tense

■ **minutesWithin** - Minutes in Range

Check if current time is between Start and End (HHMM format).

bool minutesWithin(int Start, int End);

→ bool - TRUE if in range

Example: if(minutesWithin(930, 1600))

Note: HHMM format • Handles overnight (e.g. 2200, 0200)

■ **timeOffset** - Offset Timezone

Returns offset in hours from UTC for a specific timezone.

var timeOffset(string Zone);

→ var - Hours from UTC (can be fractional)

Example: var offset = timeOffset("EST");

Notes: Zones: EST, PST, CET, etc. • Consider DST if applicable

4. CHAPTER 4 - DATA STRUCTURES

4.1. 📝 Strings

■ **strcat** - String Concatenation

Concatenate Source to the end of Dest. Modify Dest in-place. Return pointer to Dest.

string strcat(string Dest, string Source);

→ string - Concatenated Dest pointer

Example: strcat(buffer, " world");

Note: Edit Dest • Dest must have enough space • Standard C library function

■ **strcmp** - String Comparison

Compares two strings lexicographically. Returns <0 if S1<S2, 0 if equal, >0 if S1>S2.

int strcmp(string S1, string S2);

→ int - <0, 0, >0 for comparison

Example: if(strcmp(Asset, "EUR/USD") == 0)

Note: Case-sensitive • Lexicographic comparison • 0 = equal strings

■ **strcpy** - Copy String

Copy Source to Dest, including the null terminator. Overwrites the contents of Dest.

string strcpy(string Dest, string Source);

→ string - Pointer to Dest

Example: strcpy(symbol, "SPY");

Note: Overwrites Dest completely • Dest must have space • Includes null terminator

4.2. abc Advanced Strings

■ **strcon** - Concatenate Array of Strings

Concatenates arrays of strings/numbers into a single string. Versions for CONTRACT and TRADE.

string strcon(string* Array, int Length);
string strcon(TRADE* Tr);string strcon(CONTRACT* C);

→ string - Concatenated string

Example: string result = strcon(strings, 3);

Notes: Array mode: concatenate elements • Trade/Contract mode: create string IDs • Overloaded versions

■ **strtr** - Translate Characters

Translates characters in Source: each character in From is replaced with the corresponding one in To.

string strtr(string Source, string From, string To);

→ string - New string with replacements

Example: strtr("Hello", "el", "ip"); // "Hippo"

Note: Character-by-character mapping • From and To must have the same length • Create new string

■ **stridx** - Find in List

Find Name in a comma-separated list. Returns index (0-based) or -1 if not found.

int stridx(string Name, string List);

→ int - Index in list or -1

Example: int idx = stridx("EUR", "USD,EUR,GBP"); // 1

Note: CSV List • 0-based index • -1 if not found • Case-sensitive

■ **strxid** - String from Index

Returns a string corresponding to the index in the predefined list. Inverse of stridx for asset lists.

string strxid(int Index);

→ string - Element name at Index

Example: string name = strxid(2); // third asset

Note: Use current asset list • 0-based index • Complementary to stridx

■ **strw** - String Width

Calculates the pixel width of the string when rendered. Useful for UI and panel formatting.

int strw(string Text);

→ int - Width in pixels

Example: int width = strw(labelText);

Note: Depends on current font • Useful for UI layout • Returns pixel width

■ **sftoa** - Float to ASCII

Converts a floating-point number to a string with a specified number of decimal places.

string sftoa(var Value, int Decimals);

→ string - String representation

Example: string s = sftoa(3.14159, 2); // "3.14"

Note: Controls decimal precision • Useful for formatting output • Alternatives: strf()

■ **strxc** - Extract Character

Extracts character at position Index from Source string (0-based indexing).

char strxc(string Source, int Index);

→ char - Character at Index

Example: char c = strxc("Hello", 1); // 'e'

Note: 0-based index • Returns '\0' if index is out of range • Character-level access

4.3. 🗀 SERIE

4.3.1. series - Create Time Series

Description

Create a new time series (vars) to store values over time. A time series is a dynamic array that automatically grows with historical data, keeping the latest value at position [0] and previous values at [1], [2], etc. Essential for calculating indicators, moving averages, and any analysis that requires historical data.

Syntax

vars series(var Value, int Length = 0);

Parameters

Value (var)

Current value to add to the series

Example: priceClose(), ATR(14)

Length (int) - Optional

Maximum series length. If 0 (default), it grows dynamically.

Typical values: 0 (dynamic), 100, 500

Return Value

vars - Pointer to the created time series

Example of Use

```
function run() {
BarPeriod = 60;

vars PriceSeries = series(priceClose());var CurrentPrice = PriceSeries[0];var PrevPrice = PriceSeries[1];

vars FastMA = series(SMA(PriceSeries, 20), 100);vars SlowMA = series(SMA(PriceSeries, 50), 100);

if(crossOver(FastMA, SlowMA))enterLong();}
```

Usage Notes

- Index [0] is always the most recent value
- Use fixed Length to optimize memory
- Series can be passed directly to indicators

4.3.2. shift - Shift Series

Description

Shifts the values of a time series by N positions. With a positive N, it shifts into the future; with a negative N, it shifts into the past. Useful for aligning time series with different lags. CAUTION: Shifting into the future creates look-ahead bias if not handled correctly.

Syntax

void shift(vars Series, int N, var Value);

Parameters

Series (vars)

Time series to move

Example: PriceSeries

N (int)

Number of positions. Positive = future, Negative = past

Typical values: -10 to +10

Value (var)

Value to fill the created positions

Example: 0, priceClose()

Return Value

void - Edit the series directly

Example of Use

```
function run() {
vars PriceSeries = series(priceClose());vars DelayedSeries = series(priceClose());

shift(DelayedSeries, 5, priceClose());
```

if(PriceSeries[0] > DelayedSeries[0] + 10*PIP) {enterLong();}}

Usage Notes

- Moving into the future creates look-ahead bias
- Use to align series with different lags
- The fill value must be chosen carefully

4.3.3. sprintf - Format String

Description

Formats a string according to a specific format, similar to printf but returning the string. Essential for creating custom messages, formatted logs, and dynamic filenames. Supports all standard C format specifiers: %s (string), %d (int), %f (float), and %.nf (float with n decimal places).

Syntax

string sprintf(string Format, ...);

Parameters

Format (string)

Format string with placeholder

Example: "Price: %.2f, Volume: %d"

... (variadic)

Values to be inserted in the placeholders

Example: price, volume

Return Value

string - Resulting formatted string

Example of Use

```
function run() {
string msg = sprintf("Asset: %s, Price: %.2f",Asset, priceClose());

string filename = sprintf("Log%s%d.txt",Asset, year(0)*10000+month(0)*100+day(0));

if(is(LOGFILE))printf("\n%s", sprintf("[%s] Trades: %d",strdate("%Y-%m-%d", 0),NumOpenTotal));}
```

Usage Notes

- Common specifiers: %d (int), %f (float), %s (string)
- Use to create structured and readable logs
- Great for dynamic filenames with dates

4.4. Special series

4.5. Arrays

4.6. Advanced Series & Arrays

5. CHAPTER 5: I/O & REPORTING

5.1. Output

5.2. File I/O

5.3. Advanced File I/O

5.4. Reports

5.5. Flow & Loop Control

These functions control the script's execution flow and allow you to efficiently iterate through assets, trades, and statuses. They are essential for implementing multi-asset strategies and managing complex portfolios.

5.5.1. algo - Select Algorithm

Description

Select or create a specific algorithm (strategy) within a script. Algorithms
allow you to execute multiple strategies simultaneously on the same asset or on different assets.
Each algorithm has its own set of variables AlgoVar and AlgoPtr. When algo() is called with
a name that does not exist, a new algorithm is automatically created.
The use of multiple algorithms is essential for portfolio trading and to logically separate different strategies within the same script. Each algorithm maintains separate state and can be independently managed.

Syntax

string algo(string Name);

Parameters

Name (string)
Name of the algorithm to select or create

Example: "TrendFollower", "MeanReversion", "Scalper"*

Special values: NULL or "" = default algorithm

Return Value

string - Name of the previously selected algorithm

Example of Use

```
function run()
{
// Strategy 1: Trend Following
algo("Trend");
if(ADX() > 25)
enterLong();
// Strategy 2: Mean Reversion
algo("MeanRev");
if(RSI(14) < 30)
enterLong();
// Loop through all algorithms
for(allalgos)
{
printf("\nAlgo: %s, Profit: %.2f", Algo, WinTotal);
}
}
```

Usage Notes

- Each algorithm has its own set of variables AlgoVar[0..7]
- Algorithms are automatically created on first use
- Use algo(NULL) to return to the default algorithm
- In portfolio mode, each algo-asset combination has a separate state

5.5.2. allow - Trading Permission

Description

Determines whether trading is allowed for the asset and the current algorithm. This function is used internally from Zorro to check if a new position can be opened. It can be
overridden by the user to implement custom trading control logic.
The default function returns 0 (no permission) when the system is in INITRUN or when
There is no pricing data available. It can be used to implement custom rules.
such as capital limits, volatility filters, or restrictions based on external events.

Syntax

```
int allow();
```

Parameters

Nobody

Return Value

int - 1 if trading is allowed, 0 if not allowed

Example of Use

```
// Custom allow function with volatility filter
int allow()
{
// Don't trade during initialization
if(is(INITRUN))
return 0;
// Don't trade on low volatility days
if(ATR(14) < 0.001 * priceClose())
return 0;
// Don't trade if max drawdown exceeded
if(ProfitTotal < -10000)
return 0;
// Don't trade outside market hours
if(hour() < 9 || hour() >= 16)
return 0;
return 1; // Trading allowed
}
function run()
{
if(allow() && crossOver(SMA(20), SMA(50)))
enterLong();
}
```

Usage Notes

- Automatically called before each attempt to enter position
- Can be overridden for custom trading logic
- Does not affect the closing of existing positions
- Useful for implementing circuit breakers and risk limits

5.5.3. barssince - Bars Since Condition

Description

Returns the number of bars since the last time a specific condition was true.

If the condition is true in the current bar, it returns 0. If the condition has never been true in the current bar, it returns 0.

true, returns a very large value (over 10,000).

This function is particularly useful for implementing timing conditions and for checking

How much time has passed since a specific event. This is similar to the barssince() function in other functions.

trading platforms such as AFL (AmiBroker) and Pine Script.

Syntax

int barssince(bool Condition);

Parameters

Condition (bool)

Boolean condition to monitor

Example: crossOver(series1, series2), Price > MA

The condition is evaluated at each bar

Return Value

int - Number of bars since the last occurrence of the condition (0 if true now)

Example of Use

```
function run()
{
vars Price = series(priceClose());
vars MA50 = series(SMA(Price, 50));
// Number of bars since last golden cross
int BarsFromCross = barssince(crossOver(Price, MA50));
// Enter only if golden cross is recent (max 5 bars ago)
if(BarsFromCross > 0 && BarsFromCross <= 5) {
if(!NumOpenLong)
enterLong();
}
// Check time since last significant drawdown
int BarsFromDD = barssince(ProfitTotal < -1000);
if(BarsFromDD < 20) {
// Reduce size if recent drawdown
Lots = 0.5;
}
if(is(LOGFILE))
printf("\nBars from cross: %d", BarsFromCross);
}
```

Usage Notes

- Returns 0 if the condition is true in the current bar
- Returns value > 10000 if condition never met
- Useful for implementing time-based filters
- Performance-friendly: only evaluates necessary conditions

5.5.4. valuewhen - Conditional Array Access

Description

Returns the value of a time series at the specified occurrence of a condition. It is similar to barssince but instead of returning the number of bars, it returns the actual value of the series at that point.

This feature is extremely useful for implementing strategies based on historical levels, such as trading on breakouts of previous highs/lows or to compare current prices with specific historical values.

Syntax

var valuewhen(vars Series, bool Condition, int Occurrence);
```

**Parameters**

Series (vars)

Time series from which to extract the value

*Example: series(priceClose()), series(priceHigh())*

Condition (bool)

Condition that identifies the point of interest

**Example:** crossOver(Price, MA), peak(Price)*

Occurrence (int)

Which occurrence of the condition (0 = most recent, 1 = previous, etc.)

*Typical values: 0, 1, 2*

*0 = last occurrence, 1 = second to last, 2 = third to last*

**Return Value**

var - Value of the series at the specified occurrence of the condition

**Example of Use**

```
function run()
{
vars Price = series(priceClose());
vars High = series(priceHigh());
vars Low = series(priceLow());
// Find the last local maximum (peak)
var LastPeakPrice = valuewhen(Price, peak(Price), 0);
```
```

```
var PrevPeakPrice = valuewhen(Price, peak(Price), 1);
// Breakout trading: enter when we break the last peak
if(priceClose() > LastPeakPrice * 1.02) {
if(!NumOpenLong) {
printf("\nBreakout! Last peak: %.2f", LastPeakPrice);
enterLong();
}
}
// Support/Resistance levels
var LastValleyPrice = valuewhen(Price, valley(Price), 0);
// Trailing stop based on previous valley
if(NumOpenLong > 0) {
var StopLevel = max(LastValleyPrice, priceClose() * 0.95);
TradeStopLimit = StopLevel;
}
// Pattern recognition: higher highs
var Peak1 = valuewhen(High, peak(High), 0);
var Peak2 = valuewhen(High, peak(High), 1);
if(Peak1 > Peak2 * 1.01) {
printf("\nHigher high detected");
}
}
```

Usage Notes

- Occurrence 0 = most recent, 1 = previous, etc.
- Returns 0 if the condition has never been true.
- Useful for dynamic support/resistance
- Combine with peak/valley for pattern trading

5.5.5. once - First Occurrence

Description

Returns 1 (true) only the first time it is called within a specification.
condition or context. On subsequent calls, it returns 0 (false). It is reset.
when the context changes (new asset, new algorithm, or new script run).
This function is essential for executing initialization code or triggering events.
only once, avoiding repeated executions. It is particularly useful in conditions,
loops and for setups that only need to happen on the first occurrence of an event.

Syntax

```
int once();
int once(int Period);
```

Parameters

Period (int) - Optional

Number of bars after which the flag is reset

Default: never (reset only on context change)

Example: once(100) = true every 100 bars

Return Value

int - 1 on first call in the current context, 0 thereafter

Example of Use

```
function run()
{
// Initialize once per asset
if(once()) {
printf("\n=== Initializing %s ===", Asset);
AlgoVar[0] = 0; // Reset counter
}
// Execute only on the first bar
if(once() && is(FIRSTRUN)) {
printf("\nStarting backtest from %s", strdate("%Y-%m-%d", 0));
}
// Log entry only first time condition is true
if(RSI(14) < 30) {
if(once()) { // Will print only first time RSI < 30
printf("\nOversold condition detected at bar %d", Bar);
}
if(!NumOpenLong)
enterLong();
}
// Periodic execution: once every 100 bars
if(once(100)) {
printf("\nPerformance check - Profit: %.2f", ProfitTotal);
}
// Reset state once when condition becomes true
if(crossOver(SMA(20), SMA(50))) {
if(once()) {
AlgoVar[1] = 1; // Set flag
printf("\nGolden cross detected!");
}
}
```

}

Usage Notes

- Automatically reset with asset or algorithm change
- Use inside conditions to trigger one-time events
- once(N) for periodic execution every N bars
- Useful for avoiding log spam and multiple initializations

5.5.6. call - Run Function at Event

Description

Register a function to be called automatically when specific events occur.
Trading. Events include trade openings, trade closings, ticks, and bars. This
callback mechanism allows you to separate trading logic into well-defined modules.
The registered function is automatically called by Zorro when the corresponding event occurs.
occurs. This event-driven approach makes the code more modular and easier to manage.
of complex strategies.

Syntax

void call(int Event, function Callback);

Parameters

Event (int)
Type of event to monitor
Possible values:
- CALLTICK: call at every tick
- CALLBAR: call at each bar
- CALLENTRY: call when opening a position
- CALLEXIT: called when closing a position

Callback (function)
Function to call when the event occurs
Prototype: void myFunction()
The function must not have any parameters

Return Value

void - Does not return a value

Example of Use

```
// Callback for risk management
void onEntry()
{
printf("\n[ENTRY] New position opened on %s", Asset);
printf("\n Entry Price: %.5f", TradeEntryPrice);
printf("\n Stop Loss: %.5f", TradeStopLimit);
// Custom risk management
```

```
var RiskPercent = 2.0;
var RiskAmount = Capital * RiskPercent / 100;
var StopDistance = abs(TradeEntryPrice - TradeStopLimit);
// Adjust lots based on risk
TradeLots = RiskAmount / (StopDistance / PIP);
// Log to file
fileappend("trades.log",
strf("\n%s,ENTRY,%.5f,%.0f",
Assets, TradeEntryPrice, TradeLots));
}
void onExit()
{
printf("\n[EXIT] Position closed");
printf("\n P&L: %.2f", TradeResult);
printf("\n Bars held: %d", TradeBarsPeriod);
// Track statistics
AlgoVar[0] += TradeResult; // Cumulative P&L
AlgoVar[1]++; // Trade count
fileappend("trades.log",
strf("\n%s,EXIT,%.2f,%d",
Asset, TradeResult, TradeBarsPeriod));
}
void onTick()
{
// High-frequency monitoring
if(NumOpenLong > 0) {
var CurrentProfit = TradeProfit;
if(CurrentProfit > AlgoVar[2]) { // New profit peak
AlgoVar[2] = CurrentProfit;
// Adjust trailing stop
TradeStopLimit = priceClose() - 2*ATR(14)*PIP;
}
}
}
function run()
{
// Register callbacks
call(CALLENTRY, onEntry);
```

```
call(CALLEXIT, onExit);
call(CALLTICK, onTick);
// Main trading logic
if(crossOver(SMA(20), SMA(50)))
enterLong();
}
```

Usage Notes

- Callbacks are automatically called by Zorro
- Useful for separating risk management from trading logic
- CALLTICK requires TICKS flag to work
- Callbacks must not be recursive

5.5.7. forAsset - Asset Enumeration Loop

Description

Macro to iterate through all assets in an asset list. Simplifies looping through multiple assets in portfolio strategies. This is equivalent to a for loop that calls asset() for each item in the list.

This macro is essential for portfolio trading and multi-asset strategies, allowing you to apply the same trading logic to different instruments efficiently.

Syntax

```
for(listedassets) { ... }
for(selectedassets) { ... }
```

Parameters

listed assets

Loop through all assets in the current AssetList

selectedassets

Loop through assets that match specific criteria

Return Value

N/A - This is a loop macro

Example of Use

```
function run()
{
// Simple portfolio iteration
for(listedassets)
{
printf("\n%s: Price = %.2f", Asset, priceClose());
// Apply same strategy to all assets
if(RSI(14) < 30)
enterLong();
```

```
else if(RSI(14) > 70)
enterShort();
}
// Advanced: track performance across portfolio
var TotalProfit = 0;
int WinningAssets = 0;
for(listedassets)
{
TotalProfit += WinTotal;
if(WinTotal > 0)
WinningAssets++;
}
printf("\nPortfolio Profit: %.2f", TotalProfit);
printf("\nWinning Assets: %d/%d", WinningAssets, NumAssets);
}
// Example: Sector rotation
function run()
{
assetList("ETFs.csv");
// Find the strongest asset
string BestAsset = "";
var BestMomentum = -999999;
for(listedassets)
{
var Momentum = (priceClose() / priceClose(20) - 1) * 100;
if(Momentum > BestMomentum) {
BestMomentum = Momentum;
BestAsset = Asset;
}
}
// Trade only the strongest asset
asset(BestAsset);
if(!NumOpenLong)
enterLong();
}
```

Usage Notes

- Automatically calls asset() for each iteration

- Asset variable contains the current name
- All price functions refer to the current asset
- Combine with algo() for multi-strategy portfolios

5.5.8. forStatus - Status Enumeration Loop

Description

Iterate through all the statuses (asset-algorithm combinations) in the system. Each combination of assets and algorithms has its own STATUS struct which contains specific variables like AlgoVar, trade statistics, and parameters.

This function is mainly used internally by Zorro, but can be useful for analysis advanced and customized reporting on complex portfolios.

Syntax

for(allstatus) { ... }

Parameters

None - operates on the global variable Asset and Algo

Return Value

N/A - This is a loop macro

Example of Use

```
function run()
{
// Analyze all asset-algo combinations
for(allstatus)
{
printf("\n%s-%s: Trades=%d, Profit=%.2f",
Asset, Algo, NumWinTotal + NumLossTotal, WinTotal);
// Check if combination is profitable
if(WinTotal > 1000) {
printf(" [PROFITABLE]");
}
// Access AlgoVar for this status
printf(" State: %.0f", AlgoVar[0]);
}
}
// Example: Adaptive money management
function run()
{
// Calculate total capital allocation
var TotalAllocated = 0;
```

```
for(allstatus)
{
TotalAllocated += AlgoVar[5]; // Assuming AlgoVar[5] = allocated capital
}
// Now adjust current position sizing
asset("EUR/USD");
algo("Trend");
var AvailableCapital = Capital - TotalAllocated;
Lots = AvailableCapital * 0.02 / (ATR(14) * 10000);
}
```

Usage Notes

- Mainly for internal use and advanced analysis
- Access to specific AlgoVars for each status
- Useful for portfolio rebalancing
- Each status has its own trading statistics

5.5.9. forTrade - Trade Enumeration Loop

Description

Iterates through open, pending, closed, or all trades. Provides several ways to
Enumerate trades based on their status and timing. The ThisTrade variable points to the trade.
current during iteration.
This is one of the most important functions to manage multiple locations, implement
custom exit logic, and analyze trade history.

Syntax

```
for(opentrades) { ... } // All open trades
for(currenttrades) { ... } // Open trades of the current asset/algo
for(lasttrades) { ... } // Backward through trade history
for(closedtrades) { ... } // Closed trades only
for(alltrades) { ... } // All trades (open + closed)
```

Parameters

Different macros for different types of enumeration
See syntax above

Return Value

N/A - This is a loop macro. Variable ThisTrade points to the current trade.

Example of Use

```
function run()
{
// Close all losing trades
for(opentrades)
```

```
{
if(TradeProfit < -100) {
printf("\nClosing losing trade: %.2f", TradeProfit);
exitTrade(ThisTrade);
}
}
// FIFO exit: close oldest trade first
for(currenttrades)
{
if(crossUnder(SMA(20), SMA(50))) {
exitTrade(ThisTrade);
break; // Exit only first (oldest) trade
}
}
// Analyze trade history
int WinStreak = 0;
int CurrentStreak = 0;
for(closedtrades)
{
if(TradeResult > 0)
CurrentStreak++;
else {
WinStreak = max(WinStreak, CurrentStreak);
CurrentStreak = 0;
}
}
printf("\nLongest win streak: %d", WinStreak);
// Scale out: close partial positions
for(opentrades)
{
if(TradeProfit > 200) { // Profit target reached
TradeLots *= 0.5; // Close 50%
tradeUpdate(ThisTrade);
if(once())
printf("\nScaling out 50%% at profit %.2f", TradeProfit);
}
}
}
```

```
// Advanced: Correlation-based exit
function run()
{
asset("EUR/USD");
if(crossOver(SMA(20), SMA(50)))
enterLong();
// If any GBP pair has big loss, exit EUR positions
var MaxGBPLoss = 0;
for(alltrades)
{
if(strstr(Asset, "GBP") && TradeProfit < MaxGBPLoss)
MaxGBPLoss = TradeProfit;
}
if(MaxGBPLoss < -500) {
asset("EUR/USD");
exitLong(); // Correlated currency pairs
}
}
```

Usage Notes

- Variable ThisTrade always points to the current trade in the loop
- currenttrades only considers current assets and algos
- opentrades includes all assets and algos
- Be careful not to change trades during iteration unless necessary.

5.6. Advanced system and control

These features provide advanced system control, debugging, synchronization and
Memory management. These are essential for robust development and production deployment.

5.6.1. require - Require Software Version

Description

Make sure your Zorro version is at least the specified one. If it's lower,
The script stops with an error message. This function ensures that the script
only run on compatible versions of Zorro.
It is essential to use require() when your script uses available functions or features.
only in specific versions, preventing difficult-to-diagnose runtime errors.
Syntax

```
void require(int Version);
```

Parameters
Version (int)
Minimum required full-size version
*Format: year*100 + version number*
Example: 249 = version 2.49, 300 = version 3.00
Return Value
void - Terminates script if version is insufficient
Example of Use

```
function main()
{
// Require Zorro 2.50 or higher
require(250);
// Now safe to use features from 2.50+
printf("\nUsing Zorro version: %d", version());
}
function run()
{
// Require specific version for certain features
if(is(INITRUN))
require(249); // Need 2.49 for dataParseJSON
// Parse JSON data (available from 2.49)
dataParseJSON(1, "data.json");
}
// Check version and adapt behavior
function run()
{
if(version() >= 250) {
// Use new features
printf("\nUsing enhanced features");
} else {
// Fallback to older methods
printf("\nUsing legacy methods");
}
}
```

Usage Notes
- Call at the beginning of the script in main() or INITRUN

- Prevents hard-to-debug incompatibilities
- Clearly document version requirements
- Use version() for runtime checks

debuginfo - Get Debug Information

Description

Returns debug information about the system, including details about memory, performance, and internal status. Mainly used for diagnosing problems and optimizing performance. Provides access to internal metrics that are not normally visible, helping to identify bottlenecks and memory leaks in complex strategies.

Syntax

```
var debuginfo(int Type);
```

Parameters

Type (int)

Type of information requested

Values:

- 1: Memory usage (MB)
- 2: CPU time (milliseconds)
- 3: Number of bars loaded
- 4: Number of ticks processed

Return Value

var - Required value according to type

Example of Use

```
function run()
{
if(is(EXITRUN)) {
// Print debug statistics at end
printf("\n\n=== DEBUG INFO ===");
printf("\nMemory used: %.1f MB", debuginfo(1));
printf("\nCPU time: %.0f ms", debuginfo(2));
printf("\nBars loaded: %.0f", debuginfo(3));
printf("\nTicks processed: %.0f", debuginfo(4));
}
}
// Monitor memory during execution
function run()
```

```
{
static var LastMemory = 0;
var CurrentMemory = debuginfo(1);
if(CurrentMemory > LastMemory + 10) { // 10 MB increase
printf("\nWARNING: Memory increased by %.1f MB",
CurrentMemory - LastMemory);
LastMemory = CurrentMemory;
}
}
```

Usage Notes
- Mainly used for diagnostics and optimization
- Not reliable for trading logic
- Minimal performance overhead
- Values may vary between Zorro versions

suspended - Trading Suspension Status

Description

Check or set the trading suspension state. When suspended(), no new trading is created.
position can be opened, but existing ones can be closed. Useful for implementing
circuit breakers and to temporarily disable trading.
The sleep state can be controlled manually via scripting or it can be
automatically activated by Zorro under certain conditions (e.g. broker connection problems).

Syntax

```
int suspended();
int suspended(int State);
```

Parameters

State (int) - Optional
New suspension status
0 = enable trading
1 = suspend trading
Omitted = query current state

Return Value

int - Current sleep state (0 = active, 1 = suspended)

Example of Use

```
// Simple circuit breaker
```

```
function run()
{
// Suspend trading if max daily loss exceeded
if(ProfitToday < -1000) {
if(!suspended()) {
suspended(1);
printf("\n[ALERT] Trading suspended: Daily loss limit!");
sound("alert.wav");
}
}
// Re-enable next day
if(suspended() && dow() != dow(1)) { // New day
suspended(0);
printf("\nTrading re-enabled for new day");
}
}
// Volatility-based suspension
function run()
{
var CurrentVol = Volatility(30);
var AvgVol = Volatility(252);
// Suspend if volatility too high
if(CurrentVol > AvgVol * 2) {
suspended(1);
printf("\n[WARNING] High volatility - trading suspended");
}
else if(suspended() && CurrentVol < AvgVol * 1.5) {
suspended(0);
printf("\nVolatility normalized - trading resumed");
}
// Normal trading logic
if(!suspended()) {
if(crossOver(SMA(20), SMA(50)))
enterLong();
}
}
// Emergency stop with lock file
function run()
```

```
{
// Check for emergency stop file
string StopFile = "EMERGENCYSTOP.txt";
if(filedate(StopFile) > 0) { // File exists
if(!suspended()) {
suspended(1);
printf("\n\n!!! EMERGENCY STOP ACTIVATED !!!\n");
// Close all positions
for(opentrades)
exitTrade(ThisTrade);
}
}
}
```

Usage Notes

- Prevents opening new positions but not closing them
- State persisted between runs
- Automatically suspended if broker disconnected
- Use for circuit breakers and risk management

watch - Debugging Info

Description

Displays variables and expressions in the message window for debugging. Can also stop Running the script for step-by-step debugging. One of the most useful features during development to inspect the system state.

Watch can display variables in different modes and with different levels of detail, making it easier enormously the debugging of complex strategies.

Syntax

```
void watch(string Text, ...);
```

Parameters

Text (string)

Text to display with format specifiers for variables

Format specifiers: %f (float), %d (int), %s (string)

Special prefixes:

- "!" = stop execution (debug mode)
- "#" = print to log file always
- "?" = print with next error message

... (variadic)
Variables to display according to format specifiers
Return Value
void
Example of Use

```
function run()
{
// Simple variable watch
watch("Price: %.2f, RSI: %.1f", priceClose(), RSI(14));
// Stop execution for debugging
if(NumOpenLong > 5) {
watch("!Too many positions: %d", NumOpenLong);
// Script stops here - press [Go] to continue
}
// Always log to file (even in Test mode)
watch("#Trade opened: %.2f at %.5f",
TradeLots, TradeEntryPrice);
// Debugging complex conditions
var MA20 = SMA(20);
var MA50 = SMA(50);
bool CrossCondition = crossOver(series(MA20), series(MA50));
watch("MA20: %.5f, MA50: %.5f, Cross: %d",
MA20, MA50, CrossCondition);
// Print with next error (crash diagnosis)
watch("?Last values before crash: Price=%.2f RSI=%.1f",
priceClose(), RSI(14));
}
// Advanced: Conditional debugging
function run()
{
vars Price = series(priceClose());
if(Price[0] > Price[1] * 1.05) { // Suspicious spike
watch("!Price spike detected:");
watch(" Current: %.5f", Price[0]);
watch(" Previous: %.5f", Price[1]);
watch(" Change: %.2f%%", (Price[0]/Price[1]-1)*100);
watch(" Bar: %d", Bar);
```

```
}
}
// Monitor trade execution
function run()
{
if(NumWinTotal + NumLossTotal != NumWinTotal[1] + NumLossTotal[1]) {
// New trade closed
watch("#Trade #%d closed:", NumWinTotal + NumLossTotal);
watch("# Result: %.2f", TradeResult);
watch("# Bars: %d", TradeBarsPeriod);
watch("# Profit factor: %.2f", WinTotal/abs(LossTotal));
}
}
```

Usage Notes
- Prefix "!" stops execution for step debugging
- Prefix "#" always logs (ignores Verbose level)
- Prefix "?" displays with next error
- Avoid watch() in intensive loops (performance)

invalid - Invalid Value Check

Description

Checks whether a value is invalid (NaN or Inf) which may result from mathematical operations
invalid as division by zero or square root of negative number. Critical for
prevent the propagation of invalid values through calculations.
Invalid values can cause crashes or unpredictable behavior, so it is important
identify them and manage them appropriately.

Syntax

```
bool invalid(var Value);
```

Parameters

Value (var)
Value to be verified
It can be the result of any mathematical operation

Return Value

bool - true if the value is NaN or Inf, false otherwise

Example of Use

```
function run()
{
// Safe division
var Denominator = ATR(14);
var Result;
if(Denominator != 0)
Result = priceClose() / Denominator;
else
Result = 0;
if(invalid(Result)) {
printf("\n[ERROR] Invalid calculation result");
Result = 0; // Use safe default
}
// Validate indicator values
var RSIValue = RSI(14);
if(invalid(RSIValue)) {
printf("\n[WARNING] Invalid RSI at bar %d", Bar);
return; // Skip this bar
}
// Safe indicator calculation
vars Price = series(priceClose());
var StdDev = StdDev(Price, 20);
if(invalid(StdDev) || StdDev == 0) {
printf("\n[WARNING] Invalid volatility");
return;
}
var BollingerWidth = StdDev / SMA(Price, 20);
// Validate all intermediate calculations
if(invalid(BollingerWidth)) {
printf("\n[ERROR] Invalid Bollinger calculation");
BollingerWidth = 0.02; // Use typical value
}
}
// Robust function with error handling
var SafeDivide(var Numerator, var Denominator)
{
if(Denominator == 0 || invalid(Denominator))
return 0;
```

```
var Result = Numerator / Denominator;
if(invalid(Result))
return 0;
return Result;
}
function run()
{
var Ratio = SafeDivide(priceClose(), ATR(14));
// Now safe to use
if(Ratio > 100)
enterLong();
}
```

Usage Notes

- Use after operations that may generate NaN/Inf
- Validate input from external sources (API brokers, files)
- Prefer fix0() for simple division by zero
- Minimal performance overhead

changed - Change of Value Detection

Description

Detects whether a value has changed since the previous bar. Returns the difference if

The value has changed, 0 otherwise. Useful for triggering actions only when specific variables are used.

they change, avoiding repetitive actions.

This feature is particularly useful for implementing event-driven logic and for optimizing performance by running code only when needed.

Syntax

```
var changed(var* Variable);
```

Parameters

Variable (var*)

Pointer to the variable to monitor

Must be persistent variable (static or global)

Cannot be a local or temporary variable

Return Value

var - Difference from previous value, 0 if unchanged

Example of Use

```
function run()
{
static var LastSignal = 0;
var Signal = sign(SMA(20) - SMA(50));
// Trade only on signal change
var SignalChange = changed(&LastSignal);
if(SignalChange > 0) { // Changed from negative to positive
printf("\nBullish signal!");
if(!NumOpenLong)
enterLong();
}
else if(SignalChange < 0) { // Changed from positive to negative
printf("\nBearish signal!");
exitLong();
}
LastSignal = Signal;
}
// Efficient regime detection
function run()
{
static var LastRegime = 0;
// Determine market regime
var Regime;
if(ATR(14) > ATR(50))
Regime = 1; // High volatility
else
Speed = -1; // Low volatility
// Act only on regime change
if(changed(&LastRegime)) {
printf("\nRegime changed to: %s",
Speed > 0? "HIGH VOL" : "LOW VOL");
// Adjust strategy parameters
if(Regime > 0) {
Stop = 100 * PIP; // Wider stops
TakeProfit = 300 * PIP;
} else {
Stop = 30 * PIP; // Tighter stops
```

```
TakeProfit = 90 * PIP;
}
}
LastRegime = Regime;
}
// Monitor multiple conditions
function run()
{
static var LastMA = 0;
static var LastRSI = 0;
var CurrentMA = SMA(50);
var CurrentRSI = RSI(14);
// Check what changed
if(changed(&LastMA))
printf("\nMA changed by %.5f", CurrentMA - LastMA);
if(changed(&LastRSI))
printf("\nRSI changed by %.1f", CurrentRSI - LastRSI);
// Update tracked values
LastMA = CurrentMA;
LastRSI = CurrentRSI;
}
```

Usage Notes

- Variable must be static or global (persistent)
- Pass by reference with & operator
- Manually update the variable after changed()
- Useful for event-driven logic

touch - Curve Touch Detection

Description

Detects when a curve "touches" (intersects) another curve or a specific value. Unlike of crossOver/crossUnder that require full traversal, touch() also detects contacts momentary between the curves.

This feature is useful for strategies based on dynamic supports/resistances and for detecting when the price "tests" a level without necessarily crossing it.

Syntax

```
bool touch(var Value1, var Value2);
```

Parameters
Value1 (var)
First value or series
Typically: current price
Value2 (var)
Second value or series
Typically: moving average, support/resistance
Return Value
bool - true if the values touch (equal or very close), false otherwise
Example of Use

```
function run()
{
var MA50 = SMA(50);
var MA200 = SMA(200);
// Trade on MA touch (without full crossover)
if(touch(priceClose(), MA50)) {
printf("\nPrice touched MA50!");
if(priceClose() > MA200) // But above MA200
enterLong();
}
// Support/resistance touch
var Support = dayLow(1); // Previous day low
if(touch(priceLow(), Support)) {
printf("\nPrice tested support at %.5f", Support);
if(!NumOpenLong)
enterLong(); // Buy the dip
}
// Bollinger Band touch
vars Price = series(priceClose());
var BBUpper = BBands(Price, 20, 2);
var BBLower = BBands(Price, 20, -2);
if(touch(priceHigh(), BBUpper)) {
printf("\nPrice touched upper Bollinger Band");
if(NumOpenLong > 0)
exitLong(); // Take profit
}
if(touch(priceLow(), BBLower)) {
```

```
printf("\nPrice touched lower Bollinger Band");
if(!NumOpenLong)
enterLong(); // Mean reversion
}
}
// Advanced: Multiple timeframe touch
function run()
{
// Daily support/resistance
TimeFrame = 1440; // Daily
var DailySupport = Support();
var DailyResistance = Resistance();
// Back to current timeframe
TimeFrame = 60; // Hourly
if(touch(priceLow(), DailySupport)) {
printf("\nHourly price touched daily support");
enterLong();
}
if(touch(priceHigh(), DailyResistance)) {
printf("\nHourly price touched daily resistance");
exitLong();
}
}
```

Usage Notes

- More sensitive than crossOver/crossUnder
- Can generate multiple signals (both pro and con)
- Useful for mean reversion strategies
- Combine with other filters to reduce false signals

sync - Synchronization

Description

Synchronize script execution with external events or multiple timeframes.

to coordinate actions across different timeframes or to wait for specific events before proceed.

This feature is useful for multi-timeframe strategies and to ensure that certain calculations occur only when all the necessary data is available.

Syntax

```
void sync();
```

Parameters

Nobody

Return Value

void

Example of Use

```
// Multi-timeframe synchronization
function run()
{
// Calculate daily indicators
TimeFrame = 1440; // Switch to daily
var DailyTrend = sign(SMA(20) - SMA(50));
TimeFrame = 60; // Back to hourly
sync(); // Ensure timeframes aligned
// Now safe to use both timeframes
if(DailyTrend > 0) { // Daily uptrend
if(crossOver(SMA(20), SMA(50))) // Hourly signal
enterLong();
}
}
// Ensure data availability
function run()
{
if(NumBars < 200) {
sync(); // Wait for more data
return;
}
// Now have enough bars for calculations
var LongMA = SMA(200);
// ... rest of strategy
}
```

Usage Notes

- Use with multi-timeframe strategies
- Ensures data consistency
- Minimal performance overhead

- Not necessary for single-timeframe use
lock / unlock - Code Protection
Description
Lock synchronizes the behavior of multiple instances of Zorro to prevent interference.
When one instance calls lock(), other instances wait until unlock() is called.
or [Stop] is pressed.
These features are essential when multiple instances of Zorro need to access resources
shared files, databases, or API brokers that do not support concurrent access.
Syntax

int lock();
void unlock();
```
Parameters
Nobody
Return Value
lock(): int - 1 if lock acquired, 0 if [Stop] pressed
unlock(): void
Example of Use

// Safe file writing with multiple Zorros
function run()
{
if(is(EXITRUN)) {
lock(); // Acquire lock
// Only one Zorro can write at a time
string LogFile = "sharedlog.txt";
fileappend(LogFile,
strf("\n%s,%s,%.2f,%.0f",
strdate("%Y-%m-%d %H:%M", 0),
Assets, ProfitTotal, NumTrades));
unlock(); // Release lock
}
}
// Prevent simultaneous broker API access
function run()
{
// Multiple Zorros trading same account
```

```
if(crossOver(SMA(20), SMA(50))) {
if(lock()) { // Try to acquire lock
// Check available margin before trading
brokerCommand(GETACCOUNT, &Balance);
if(Balance > 10000) {
enterLong();
}
unlock();
}
else {
// Lock failed (Stop pressed)
printf("\nCould not acquire lock - stopping");
return;
}
}
}
// Database access protection
UpdateDatabase() function
{
lock();
// Safe database operations
string Query = strf(
"INSERT INTO trades VALUES ('%s','%s',%.2f)",
Asset, Algo, TradeResult);
// Execute query (pseudo-code)
// dbexecute(Query);
unlock();
}
function run()
{
if(NumWinTotal + NumLossTotal > (NumWinTotal + NumLossTotal)[1]) {
// Trade closed
UpdateDatabase();
}
}
// Timeout example
function run()
{
```

```
if(lock()) {
// Protected code section
timer(); // Start timer
// Critical operation
var Result = SomeLongOperation();
var Elapsed = timer();
printf("\nOperation took %.0f ms", Elapsed);
unlock();
}
}
```

Usage Notes

- Zorro S only (commercial version)
- Use for shared resources (files, broker API)
- Always unlock() after lock()
- lock() can return 0 if [Stop] is pressed
- Avoid deadlocks: always unlock even in case of errors

memory - Memory Allocation Info

Description

Returns information about the memory allocated by the system. Useful for diagnosing memory leaks and to optimize memory usage in strategies that handle large amounts of data.

Monitoring memory usage is critical for portfolio strategies with many assets and for long backtests that may exhaust available memory.

Syntax

```
var memory(int Type);
```

Parameters

Type (int)

Type of information requested

0 = Total allocated memory (bytes)

1 = Peak memory usage (bytes)

2 = Available free memory (bytes)

Return Value

var - Requested value in bytes (convert to MB with /1048576)

Example of Use

```
function run()
{
// Monitor memory usage
static var MaxMemory = 0;
var CurrentMemory = memory(0) / 1048576; // Convert to MB
if(CurrentMemory > MaxMemory) {
MaxMemory = CurrentMemory;
printf("\nPeak memory: %.1f MB", MaxMemory);
}
// Warn if memory too high
if(CurrentMemory > 1000) { // 1 GB
printf("\n[WARNING] High memory usage: %.1f MB", CurrentMemory);
}
}
// Memory profiling
function main()
{
var InitialMemory = memory(0);
printf("\nInitial memory: %.1f MB", InitialMemory/1048576);
}
function run()
{
// Your strategy code here
}
function exitRun()
{
var FinalMemory = memory(0);
var PeakMemory = memory(1);
var AvailableMemory = memory(2);
printf("\n\n=== MEMORY REPORT ===");
printf("\nFinal: %.1f MB", FinalMemory/1048576);
printf("\nPeak: %.1f MB", PeakMemory/1048576);
printf("\nAvailable: %.1f MB", AvailableMemory/1048576);
}
// Adaptive data management
function run()
{
var AvailableMB = memory(2) / 1048576;
```

```
if(AvailableMB < 500) { // Less than 500 MB free
printf("\n[WARNING] Low memory - reducing dataset");
// Free up some memory
dataDelete(1);
dataDelete(2);
// Reduce asset list
NumAssetsMax = 50; // Trade fewer assets
}
}
// Memory leak detection
function run()
{
static var LastMemory = 0;
static int BarCounter = 0;
BarCounter++;
if(BarCounter % 1000 == 0) { // Check every 1000 bars
var CurrentMemory = memory(0);
if(LastMemory > 0) {
var MemoryIncrease = (CurrentMemory - LastMemory) / 1048576;
if(MemoryIncrease > 10) { // More than 10 MB increase
printf("\n[LEAK?] Memory increased by %.1f MB in 1000 bars",
MemoryIncrease);
}
}
LastMemory = CurrentMemory;
}
}
```

Usage Notes

- Values in bytes - divide by 1048576 per MB
- Windows 32-bit: max 3 GB per process
- Use 64-bit Zorro for large datasets
- Monitor periodically to detect leaks

5.7. DATA MANAGEMENT

These functions handle generic datasets, allowing you to load, parse, manipulate and Save data in various formats. Essential for importing external data, creating custom indicators, and implement machine learning strategies.

5.7.1. dataDownload - Download Market Data

Description

Download datasets from Quandl™ or other online data sources and save them in CSV format in the History folder. Data is downloaded only if it is more recent than the existing data plus the specified period.

This feature enables complete automation of data management, automatically downloading updated data when necessary to keep historical databases always current.

Syntax

int dataDownload(int Code, string Source, int Period);

Parameters

Code (int)

Identification code of the dataset to download

Values: 1-999 (IDs for datasets)

Some predefined codes available in the documentation

Source (string)

Dataset source (e.g. Quandl code)

Quandl format: "VENDOR/SYMBOL"

Example: "CHRIS/CMECL1" (Crude Oil futures)

Period (int)

Minimum period in minutes between downloads

0 = always download

1440 = once a day

10080 = once a week

Return Value

int - Number of records downloaded, 0 if error or not needed

Example of Use

```
// Download Crude Oil futures data
function main()
{
// Download once per day (1440 minutes)
int Records = dataDownload(1, "CHRIS/CMECL1", 1440);
if(Records > 0)
printf("\nDownloaded %d records of CL futures", Records);
else
printf("\nUsing cached data or download failed");
// Parse the downloaded CSV
dataParse(1, "%Y-%m-%d,f3,f1,f2,f4,f6,f5", "History\\CMECL1.csv");
dataSave(1, "History\\CL.t6");
```

```
}
// Download multiple assets
function main()
{
string Symbols[] = {
"CHRIS/CMECL1", // Crude Oil
"CHRIS/CMENG1", // Natural Gas
"CHRIS/CMEGC1" // Gold
};
for(i=0; i<3; i++)
{
printf("\nDownloading %s...", Symbols[i]);
int Records = dataDownload(i+1, Symbols[i], 1440);
if(Records > 0) {
printf(" OK (%d records)", Records);
}
else {
printf(" Cached or Failed");
}
}
}
// COT report download (Commitment of Traders)
var CFTCSP(int Column)
{
dataDownload(802, "CFTC/TIFFCMESPALL", 10080); // Weekly
return dataFromQuandl(802,
"%Y-%m-%d,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f",
"CFTC/TIFFCMESPALL",
Column);
}
function run()
{
// Use COT data in strategy
var CommercialLongs = CFTCSP(3);
var CommercialShorts = CFTCSP(4);
if(CommercialLongs > CommercialShorts)
enterLong();
}
```

```
```

Usage Notes

- Requires Zorro S for Quandl access
- File saved in History folder
- Period 0 = always download (for debugging)
- Always check the return value
- Some services require API keys

5.7.2. dataParse - Parse CSV to Dataset

Description

Parses records from a CSV file and appends them to the beginning of the specified dataset. Supports time/date, floating point, integer and text fields. The format string determines how to interpret each field in the CSV.

This is one of the most important functions for importing external data into Zorro, with great flexibility in handling different CSV formats.

Syntax

```
int dataParse(int Handle, string Format, string Filename);
int dataParse(int Handle, string Format, string Filename, int Start, int Num);
```

Parameters

Handle (int)

Dataset handle (1-999)

Unique identifier for this dataset

Format (string)

Format string for parsing

Describes the structure of the CSV

Full details in the Format String section

Filename (string)

Name of the CSV file to parse

Relative or absolute path

Ex: "History\\data.csv"

Start (int) - Optional

Record to start from (0-based)

Default: 0 (file start)

Num (int) - Optional

Number of records to parse

Default: 0 (all records)

Return Value

int - Number of records parsed, 0 if error
Format String Syntax
Format: "DateFormat,field1,field2,field3,..."
Date Formats:
%Y Year (4 digits) 2025
%y Year (2 digits) 25
%m Month (01-12) 03
%d Day (01-31) 15
%H Hour (00-23) 14
%M Minute (00-59) 30
%S Second (00-59) 45
%t Unix timestamp 1234567890
Field Types:
f Float field (var) 123.456
fn Float stored in column n
i Integer field 12345
s String field "TEXT"
, Skip this field
|| Skip multiple fields
Prefixes:
+ Start new dataset (clear before parse)
- Parse in reverse order
```

**Example of Use**

```
// Quandl futures format to .t6
function main()
{
// Format: Date,Open,High,Low,Settle,Volume,OpenInt
// Target: Date,Open,High,Low,Close,Volume,fVal
string Format = "%Y-%m-%d,f3,f1,f2,f4,,,f6,f5";
dataParse(1, Format, "History\\CL1.csv");
dataSave(1, "History\\CL.t6");
}
// Yahoo data format
function main()
{
// Yahoo: Date,Open,High,Low,Close,AdjClose,Volume
// Store AdjClose in fVal
```
```

```
string Format = "%Y-%m-%d,f3,f1,f2,f4,f6,f5";
dataParse(1, Format, "History\\AAPL.csv");
dataSave(1, "History\\AAPL.t6");
}
// Bitcoin tick data
function main()
{
// Format: timestamp,price,volume,buysell
string Format = "+0,,,%t,f,f,s";
int Records = dataParse(1, Format, "History\\BTCEUR.csv");
printf("\n%d records read", Records);
// Convert to t1 format
dataNew(2, 0, 0);
for(i=0; i<Records; i++)
{
T1* t1 = dataAppendRow(2, 2);
t1->time = dataVar(1, i, 0);
var Price = dataVar(1, i, 1);
string BuySell = dataStr(1, i, 2);
// Negative for bid, positive for ask
t1->fVal = (BuySell[0] == 'B') ? Price : -Price;
}
dataSave(2, "History\\BTCEUR.t1");
}
// Parse only recent data
function main()
{
string Format = "%Y-%m-%d,%H:%M:%S,f,f";
// Parse last 1000 records
int Total = filelength("History\\ticks.csv");
int Records = dataParse(1, Format, "History\\ticks.csv",
max(0, Total-1000), 1000);
printf("\nParsed %d of %d records", Records, Total);
}
// Multiple symbols in same file
function main()
{
string Symbols[] = {"AAPL", "GOOGL", "MSFT"};
```

```
string Format = "%Y-%m-%d,s,f,f,f,f,f"; // s = symbol field
dataParse(1, Format, "History\\multisymbol.csv");
// Split by symbol
for(s=0; s<3; s++)
{
dataNew(s+10, 0, 0);
int Rows = dataSize(1);
for(i=0; i<Rows; i++)
{
string Symbol = dataStr(1, i, 1);
if(strcmp(Symbol, Symbols[s]) == 0) {
// Copy this row to symbol-specific dataset
dataAppend(s+10, 1, i, i+1);
}
}
dataSave(s+10, strf("History\\%s.t6", Symbols[s]));
}
}
```

Usage Notes

- CSV headers are automatically skipped
- Use + prefix to clear dataset before parse
- Use - prefix for reverse chronological order
- Check with dataSaveCSV() to debug format string
- Large files: set Verbose >= 2 for progress dots

5.7.3. dataParseJSON - Parse JSON to Dataset

Description

Parse OHLC data from a JSON file and create a .t6 dataset. Supports various JSON formats.
common used by exchanges and data providers for price data.
This feature simplifies the import of data from modern REST APIs which typically
return data in JSON format rather than CSV.

Syntax

```
int dataParseJSON(int Handle, string Filename);
int dataParseJSON(int Handle, string Filename, string KeyOpen, string KeyHigh,
string KeyLow, string KeyClose, string KeyVol, string KeyTime);
```

Parameters

Handle (int)
Dataset handle (1-999)
Filename (string)
Name of the JSON file to parse
KeyOpen, KeyHigh, etc. (string) - Optional
JSON keys for OHLCVT fields
If omitted, use common standard keys
Return Value
int - Number of records parsed, 0 if error
Example of Use

```
// Simple JSON parsing (standard keys)
function main()
{
int Records = dataParseJSON(1, "History\\btcohlc.json");
if(Records > 0) {
printf("\nParsed %d OHLC records", Records);
dataSave(1, "History\\BTCUSD.t6");
}
}
// Custom JSON keys
function main()
{
// JSON with non-standard field names
int Records = dataParseJSON(1, "History\\crypto.json",
"openingprice", // Open
"highest price", // High
"lowestprice", // Low
"closing price", // Close
"tradevolume", // Volume
"timestamp"); // Time
printf("\nParsed %d records", Records);
}
// Download and parse in one step
function main()
{
// Download from REST API
string URL = "https://api.exchange.com/candles?symbol=BTCUSD";
```

```
string JSON = httptransfer(URL, 0);
if(JSON) {
filewrite("History\\temp.json", JSON, 0);
int Records = dataParseJSON(1, "History\\temp.json");
if(Records > 0) {
dataSave(1, "History\\BTCUSD.t6");
printf("\nConverted %d candles to t6 format", Records);
}
}
}
// Parse multiple timeframes
function main()
{
string Timeframes[] = {"1m", "5m", "1h", "1d"};
for(i=0; i<4; i++)
{
string URL = strf(
"https://api.exchange.com/ohlc?interval=%s",
Timeframes[i]);
string JSON = httptransfer(URL, 0);
filewrite("History\\temp.json", JSON, 0);
dataParseJSON(i+1, "History\\temp.json");
dataSave(i+1, strf("History\\BTC%s.t6", Timeframes[i]));
}
}
```

Usage Notes

- Output format is always .t6 (OHLCV + fVal)
- Supports nested JSON with dot notation
- Timestamps can be Unix or ISO format
- For complex JSON, use custom parsing with sscanf

5.7.4. dataParseString - Parse String to Dataset

Description

Parse a string containing CSV-formatted data directly into a dataset, without the need for an intermediate file. Useful when the data comes from HTTP APIs, database queries, or is programmatically generated.

Syntax

```
int dataParseString(int Handle, string Format, string Data);
```

Example of Use

```
string Response = httptransfer("https://api.forex.com/rates", 0);
int Records = dataParseString(1, "%Y-%m-%d,f,f,f,f", Response);
```

5.7.5. dataSave / dataSaveCSV - Save Dataset

Description

dataSave saves a dataset in binary format (.t6/.t1). dataSaveCSV saves in readable CSV format.

Syntax

```
int dataSave(int Handle, string Filename);
int dataSaveCSV(int Handle, string Format, string Filename);
```

5.7.6. dataLoad - Load Dataset

Description

Load a dataset saved from a binary file.

Syntax

```
int dataLoad(int Handle, string Filename, int Mode);
```

5.7.7. Remaining Functions

The following features are documented in the official Zorro documentation:

- dataNew: Create a new empty dataset
- dataSort: Sort dataset by timestamp
- dataMerge: Merges two datasets
- dataAppend: Add records to dataset
- dataAppendRow: Appends a single row
- dataFind: Find records by date
- dataSet: Set field value
- dataVar/dataInt/dataStr: Reads values from dataset
- dataSize: Gets the dataset size
- dataCompress: Remove duplicates
- dataClip: Remove records
- dataCopy: Copy dataset
- dataDelete: Delete dataset
- dataRearrange: Reorder fields

- dataCol: Extract column
- dataRow: Gets row number
- dataChart: View dataset

6. CHAPTER 6: OPTIONS.C - OPTIONS TRADING LIBRARY

6.1. BLACK-SCHOLES PRICING & GREEKS

6.1.1. BSPrice

Calculate the theoretical price of an option using the Black-Scholes-Merton model.

```
  var BSPrice(
int Type, // CALL (1) or PUT (2)
var S, // Spot price
var X, // Strike price
var T, // Time to expire (years)
var r, // Risk-free rate (annual)
var d, // Dividend yield (annual)
   var v // Volatility (annual)
);
```

Example:

```
  var callPrice = BSPrice(CALL, 4500, 4550, 30/365.0, 0.05, 0.01, 0.20);
var putPrice = BSPrice(PUT, 4500, 4450, 30/365.0, 0.05, 0.01, 0.20);
printf("\nCall: $%.2f Put: $%.2f", callPrice, putPrice);
```

6.1.2. BSDelta

Calculate the theoretical Delta (sensitivity to the movement of the underlying).

```
  var BSDelta(int Type, var S, var X, var T, var r, var d, var v);
```

Example:

```
var callDelta = BSDelta(CALL, 4500, 4550, 30/365.0, 0.05, 0.01, 0.20);
  var putDelta = BSDelta(PUT, 4500, 4450, 30/365.0, 0.05, 0.01, 0.20);
// Call delta ~ +0.30, Put delta ~ -0.30
```

6.1.3. BSGamma

Calculates the Gamma (Delta's sensitivity to spot movement). Identical for CALL and PUT.

```
var BSGamma(var S, var X, var T, var r, var d, var v);
```

Example:

```
var gamma = BSGamma(4500, 4500, 30/365.0, 0.05, 0.01, 0.20);
printf("\nGamma: %.4f", gamma);
```

6.1.4. BSVega

Calculate Vega (sensitivity to implied volatility). Identical for calls and puts.

```
var BSVega(var S, var X, var T, var r, var d, var v);
```

6.1.5. BSTheta

Calculate Theta (time decay, per day).

```
var BSTheta(int Type, var S, var X, var T, var r, var d, var v);
```

Example:

```
var callTheta = BSTheta(CALL, 4500, 4500, 30/365.0, 0.05, 0.01, 0.20);
printf("\nTheta: $%.2f/day", callTheta); // Typically negative for LONG
```

6.1.6. BSRho

Calculate the Rho (sensitivity to the risk-free rate).

```
var BSRho(int Type, var S, var X, var T, var r, var d, var v);
```

6.1.7. BSImplVol

Calculate Implied Volatility using the Newton-Raphson method.

```
var BSImplVol(
int Type, // CALL or PUT
var Price, // Observed price
var S, // Spot price
var X, // Strike
var T, // Time to expire (years)
var r, // Risk-free rate
var d // Dividend yield
);
```

Example:

```
var observedPrice = 25.50;
var iv = BSImplVol(CALL, observedPrice, 4500, 4500, 30/365.0, 0.05, 0.01);
printf("\nImplied Volatility: %.2f%%", iv * 100);
```

6.2. GREEK LIVE TRADING

6.2.1. getContractGreeks

Get Greeks for live trading via brokerCommand(GETGREEKS).

```
int getContractGreeks(CONTRACT* C, var* greeks);
```

Parameters:

- C: Pointer to the contract
- greeks: Array[5] to receive [delta, gamma, vega, theta, rho]

Return: 1 if successful, 0 if unsuccessful

Example:

```
CONTRACT C;
contract(&C, CALL, strike, dte);

var greeks[5];
if(is(LIVE)) {
if(getContractGreeks(&C, greeks)) {
var delta = greeks[0];
var gamma = greeks[1];
var vega = greeks[2];
var theta = greeks[3];
var rho = greeks[4];
   }
} else {
// In BACKTEST use global variables
var delta = ContractDelta;
var gamma = ContractGamma;
}
```

IMPORTANT NOTE: The Greeks from GETGREEKS may differ slightly from those calculated by Zorro in backtesting. Always use ContractDelta, ContractGamma, etc. in backtesting.

6.2.2. updateOPTCONTRACTGreeks

Update greeks in OPTCONTRACT structure (automatically handles backtest vs live).

```
  int updateOPTCONTRACTGreeks(CONTRACT* C, OPTCONTRACT* opt);
```

Example:

```
OPTCONTRACT opt;
CONTRACT C;
contract(&C, CALL, strike, dte);
contractFromZorro(&C, &opt); // Copy basic data
updateOPTCONTRACTGreeks(&C, &opt); // Update Greeks
```

6.3. CONTRACT MANAGEMENT

6.3.1. contractFromZorro

Create OPTCONTRACT snapshots from Zorro global variables.

```
  OPTCONTRACT* contractFromZorro();
```

Example:

```
contract(CALL, 4500, 45);
OPTCONTRACT* opt = contractFromZorro();
printf("\nStrike: %.0f Delta: %.3f", opt->fStrike, opt->fDelta);
```

6.3.2. contractUpdateFromZorro

Update existing OPTCONTRACT with current data.

```
void contractUpdateFromZorro(OPTCONTRACT* C);
```

6.3.3. enterContract

Optimized entry at mid-price with progressive attempts (reduces slippage 50-80%).

```
TRADE* enterContract(
int Num, // Number of contracts (negative = sell, positive = buy)
var Limit, // Initial limit price (0 = use automatic mid)
   int Steps, // Number of steps from limit to market price
   var Wait // Fill wait time (milliseconds)
);
```

Example:

```
contract(CALL, 4500, 45);
// Mid-price entry with 5 progressive attempts
if(enterContract(-1, 0, 5, 10000)) {
printf("\nâœ… Short Call sold @ $%.2f", contractMid());
}
```

Advantages vs enterShort():

- âœ… Entry at mid price instead of bid (sell) / ask (buy)
- âœ… Slippage reduction 50-80% compared to market orders
- âœ… Progressive attempts until a guaranteed fill

6.3.4. contractMid

Calculate mid price (bid + ask) / 2.

```
var contractMid();
```

Example:

```
contract(CALL, 4500, 45);
var mid = contractMid();
printf("\nBid: $%.2f | Mid: $%.2f | Ask: $%.2f",
     ContractBid, mid, ContractAsk);
```

6.3.5. contractSpreadPct

Calculate the bid-ask spread as a percentage of the mid.

```
var contractSpreadPct();
```

Example:

```
contract(CALL, 4500, 45);
var spread = contractSpreadPct();
if(spread > 10) {
```

```
printf("\nâš ï¸ High spread: %.2f%%", spread);
  }
```

6.3.6. isContractLiquid

Check if the contract is sufficiently liquid.

```
bool isContractLiquid(
var minVolume, // Minimum volume required
var maxSpreadPct // Maximum acceptable spread (%)
);
```

Example:

```
contract(CALL, 4500, 45);
if(!isContractLiquid(100, 10)) {
printf("\nâ Œ Illiquid contract:");
printf("\n Volume: %.0f (min: 100)", ContractVol);
printf("\n Spread: %.2f%% (max: 10%%)", contractSpreadPct());
   return;
}
```

Advantages:

- âœ… Avoid illiquid contracts (-30% real slippage)
- âœ… Automatically filter out low-quality contracts

6.3.7. contractLog

Print detailed information about the current contract (debug).

```
  void contractLog(string prefix);
```

Example:

```
contract(CALL, 4500, 45);
contractLog("ENTRY CALL");

// Output:
// ENTRY CALL OPTCONTRACT INFO:
// â”œâ”€ Strike: $4500
// â”œâ”€ Price: $45.50 (Bid: $45.00 | Ask: $46.00)
// â”œâ”€ DTE: 45 days
// â”œâ”€ Greeks:
// â”‚  â”œâ”€ Delta: 0.520
// â”‚  â”œâ”€ Range: 0.0028
// â”‚  â”œâ”€ Theta: $-12.30/day
// â”‚  â””â”€ Vega: 6.50pm
```

// â”œâ”€ Volume: 1250
// â”œâ”€ Open Interest: 0
// â””â”€ Spread: $1.00 (2.20%)

6.3.8. findStrikeNearest

Find available strikes closest to target.

```
  var findStrikeNearest(
int Type, // CALL or PUT
var Target, // Target Price
var Days // DTE target
);
```

Example:

```
var targetPrice = 4525;
var nearest = findStrikeNearest(CALL, targetPrice, 45);
printf("\nNearest strike to $%.2f: $%.0f", targetPrice, nearest);
```

6.4. STRIKE FINDING

6.4.1. findStrikeByDelta

Find strikes with specific delta.

```
  var findStrikeByDelta(
int Type, // CALL or PUT
   var Delta, // Delta target (0.0-1.0, absolute value)
   int DTE, // Days to expire
var StrikeStep // Strike increment (5, 25, 50, etc.)
);
```

Example:

```
var dte = findMaturity(45);
var strikeCall = findStrikeByDelta(CALL, 0.30, (int)dte, 25);
var strikePut = findStrikeByDelta(PUT, 0.30, (int)dte, 25);

contract(CALL, strikeCall, (int)dte);
printf("\nCALL Strike: %.0f Delta: %.3f", strikeCall, ContractDelta);
```

Note:

- Delta always as an absolute value (0.30, not -0.30)
- Optimized function with O(log n) binary search

6.4.2. findStrikeOTM

Find OTM strikes at a specific percentage distance.

```
  var findStrikeOTM(
int Type, // CALL or PUT
   var PctOTM, // OTM percentage (e.g. 5.0 = 5%)
   var StrikeStep // Strike increment
);
```

Example:

```
var currentPrice = priceClose();
var strikePut = findStrikeOTM(PUT, 5.0, 25); // 5% OTM PUT
printf("\nSpot: $%.2f 5%% OTM PUT Strike: $%.0f",
      currentPrice, strikePut);
```

6.5. MATURITY

6.5.1. findMaturity

Find first available expiration >= minimum days.

```
  var findMaturity(int minDays);
```

Example:

```
  var dte = findMaturity(30); // At least 30 days
if(dte == 0) {
printf("\nâ Œ No expiration available");
return;
}
printf("\nExpiration date found: %.0f days", dte);
```

6.6. AGGREGATED GREEKS

Functions to calculate net Greeks of multi-leg strategies.

6.6.1. Array Preparation

```
OPTCONTRACT* legs[4];

// Populate with contracts
contract(CALL, 4550, 45);
legs[0] = contractFromZorro(); // Short call high

contract(CALL, 4525, 45);
legs[1] = contractFromZorro(); // Long low call

contract(PUT, 4475, 45);
legs[2] = contractFromZorro(); // Long put high
```

```
contract(PUT, 4450, 45);
legs[3] = contractFromZorro(); // Short put low
```

6.6.2. comboDelta

Calculate the strategy's net delta.

```
var comboDelta(OPTCONTRACT Legs, int NumLegs);
```

Example:

```
var netDelta = comboDelta(legs, 4);
printf("\nðŸ“Š Iron Condor Net Delta: %.3f", netDelta);

// Target: ~0.0 (delta neutral)
if(abs(netDelta) < 0.10)
printf("\nâœ… Delta neutral OK");
else
printf("\nâš ï¸ Unbalanced: delta %.3f", netDelta);
```

6.6.3. comboGamma

Calculate Net Gamma.

```
var comboGamma(OPTCONTRACT Legs, int NumLegs);
```

Interpretation:

- Gamma < 0: Short options (suffers rapid movements)
- Gamma > 0: Long options (benefits from movement)

6.6.4. comboTheta

Calculate Net Theta (daily decay).

```
var comboTheta(OPTCONTRACT Legs, int NumLegs);
```

Interpretation:

- Theta > 0: Gains from time (short net)
- Theta < 0: Lose from time (long net)

Example:

```
var netTheta = comboTheta(legs, 4);
printf("\nâ ° Daily Theta: $%.2f/day", netTheta);

if(netTheta > 0)
printf("\nâœ… Time decay in favor: $%.2f/day", netTheta);
```

6.6.5. comboVega

Calculate Net Vega (IV sensitivity).

var comboVega(OPTCONTRACT Legs, int NumLegs);

Interpretation:

- Vega < 0: Suffers IV spike (short options)
- Vega > 0: Benefits spike IV (long options)

6.7. ANALYTICS & VALIDATION

6.7.1. validateGreeks

Compare Zorro vs. theoretical Black-Scholes (QA/debugging).

void validateGreeks(var IV, var RiskFreeRate);

Example:

```
contract(CALL, 4500, 45);
validateGreeks(0.18, 0.05); // IV=18%, rate=5%

// Automatic output:
// â•”â• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to•-
// â•' GREEKS VALIDATION (BS) â•'
// â• â• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• â•£
// â•' Price: Zorro: $45.50 | BS: $45.45
// â•' Delta: Zorro: 0.520 | BS: 0.519
// â•' Gamma: Zorro: 0.0028 | BS: 0.0028
// â•' Theta: Zorro: $-12.30| BS: $-12.25
// â•' Vega: Zorro: 18.50 | BS: 18.48
// â•šâ• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â• â•
```

Interpretation of differences:

- < 1%: Normal (numerical approximations)
- 1-5%: Acceptable (rounding)
- 5%: âš ï¸ Warning! (corrupt data, incorrect IV)

6.7.2. avgWeeklyRange

Calculate weekly average range (for position sizing).

var avgWeeklyRange(int weeks); // Default: 52 weeks

Example:

```
var avgRange = avgWeeklyRange(26); // Last 6 months
var strikeWidth = avgRange * 0.10; // Width = 10% of range
printf("\nAvg weekly range: $%.2f", avgRange);
printf("\nSuggested spread width: $%.2f", strikeWidth);
```

6.7.3. optionsinfo

Print library info banner (version, features).

```
void optionsinfo();
```

Example:

```
function run()
{
if(is(INITRUN))
optionsinfo();
}

// Output:
// â•”â• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to•-
// â•' options.c - Version 2.3.0 â•'
// â•' Complete Options for Zorro S/X â•'
// â• â• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• â•£
// â•' Features: â•'
// â•' âœ… Bid/Ask aware entry â•'
// â•' âœ… Black-Scholes validation â•'
// â•' âœ… Liquidity filtering â•'
// â•' âœ… Multi-leg greeks aggregation â•'
// â•šâ• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to•
```

6.7.4. getUnderlyingPosition

Returns net position on the underlying (for hedging Hedge=2).

```
int getUnderlyingPosition(); // Net lots of futures/stocks
```

Example:

```
int futPos = getUnderlyingPosition();
if(futPos != 0) {
printf("\nâš ï¸ Future hedge position: %d lots", futPos);
  }
```

6.8. UTILITIES

6.8.1. isTradeOpen

Check if specific trade is still open.

```
int isTradeOpen(TRADE* Ticket); // 1 = open, 0 = closed
```

Example:

```
TRADE* myTrade = enterLong();
// ... later ...
if(isTradeOpen(myTrade)) {
printf("\nTrade still open");
  }
```

6.8.2. isUSHoliday

Check if the date is a US holiday (markets closed).

```
  int isUSHoliday(var checkDate, int* Holidays);
```

Example:

```
if(isUSHoliday(wdate(), NULL)) {
printf("\nðŸš« Market closed - US Holiday");
   return;
}
```

7. CHAPTER 7: HELPERS.C - UTILITY LIBRARY

7.1. STRIKE UTILITIES

7.1.1. roundStrikeToIncrement

Round price to the nearest multiple.

```
  var roundStrikeToIncrement(var price, var increment);
```

Examples:

```
var strike5 = roundStrikeToIncrement(4567.34, 5); // 4565
var strike25 = roundStrikeToIncrement(4567.34, 25); // 4575
  var strike50 = roundStrikeToIncrement(4567.34, 50); // 4550
```

7.1.2. findATMStrike

Wrapper for roundStrikeToIncrement (name clarity).

```
  var findATMStrike(var currentPrice, var increment);
```

Example:

```
var spot = priceClose();
var atmStrike = findATMStrike(spot, 25);
printf("\nSpot: $%.2f ATM Strike: $%.0f", spot, atmStrike);
```

7.1.3. offsetStrike

Calculate strike offset from strike base (for spreads).

```
var offsetStrike(var baseStrike, var offset, var increment);
```

Example:

```
var strikeBase = 5000;
var strikeOTMput = offsetStrike(strikeBase, -50, 5); // 4950
var strikeITMcall = offsetStrike(strikeBase, +30, 5); // 5030
```

7.1.4. getStrikeStep

Gets effective strike step per contract.

```
var getStrikeStep(CONTRACT* C);
```

Note: Many options have variable strike steps:

- SPX: $5 near ATM, $25 very OTM
- SPY: $1 near ATM, $5 away

7.1.5. getStrikeStepAtDistance

Gets strike step at specific distance from spot.

```
  var getStrikeStepAtDistance(string Symbol, var targetStrike);
```

Example:

```
var spotPrice = priceClose();
var targetStrike = spotPrice - 100;
var step = getStrikeStepAtDistance("SPX", targetStrike);
printf("\nStrike step @ $%.0f: $%.0f", targetStrike, step);
```

7.2. GREEKS CALCULATIONS

7.2.1. NETGREEKS Structure

Structure for aggregate Greeks.

```
typedef struct {
var delta; // Net Delta
var gamma; // Net Gamma
var theta; // Net Theta (per day)
var vega; // Net Vega
var rho; // Net Rho
var premium; // Net Premium (positive credit, negative debit)
} NETGREEKS;
```

7.2.2. calculateNetGreeksStraddle

Calculate net greeks for STRADDLE.

```
NETGREEKS calculateNetGreeksStraddle(
var callPremium, var callDelta, var callGamma, var callTheta, var callVega,
var putPremium, var putDelta, var putGamma, var putTheta, var putVega,
   int isLong, // 1 = LONG straddle, -1 = SHORT straddle
```

```
   var quantity // Number of straddles
);
```

Example:

```
contract(CALL, strike, dte);
  var callPremium = ContractPrice;
var callDelta = ContractDelta;
var callGamma = ContractGamma;
var callTheta = ContractTheta;
var callVega = ContractVega;

contract(PUT, strike, dte);
var putPremium = ContractPrice;
  var putDelta = ContractDelta;
var putGamma = ContractGamma;
var putTheta = ContractTheta;
var putVega = ContractVega;

NETGREEKS ng = calculateNetGreeksStraddle(
callPremium, callDelta, callGamma, callTheta, callVega,
   putPremium, putDelta, putGamma, putTheta, putVega,
   -1, // SHORT straddle
1 // 1 straddle
);

printf("\nNet Premium: $%.2f (CREDIT)", ng.premium);
  printf("\nNet Delta: %.3f", ng.delta);
  printf("\nNet Theta: $%.2f/day", ng.theta);
```

7.2.3. calculateNetGreeksSpread

Calculate net greeks for vertical spread.

```
NETGREEKS calculateNetGreeksSpread(
var shortPremium, var shortDelta, var shortGamma, var shortTheta, var shortVega,
   var longPremium, var longDelta, var longGamma, var longTheta, var longVega,
var quantity
);
```

7.2.4. calculateNetGreeksCondor

Calculate net greeks for IRON CONDOR.

```
NETGREEKS calculateNetGreeksCondor(
var shortCallPremium, var shortCallDelta, var shortCallGamma,
var shortCallTheta, var shortCallVega,
var longCallPremium, var longCallDelta, var longCallGamma,
var longCallTheta, var longCallVega,
var shortPutPremium, var shortPutDelta, var shortPutGamma,
var shortPutTheta, var shortPutVega,
var longPutPremium, var longPutDelta, var longPutGamma,
   var longPutTheta, var longPutVega,
var quantity
);
```

7.2.5. calculateNetGreeksButterfly

Calculate net greeks for BUTTERFLY.

```
NETGREEKS calculateNetGreeksButterfly(
var wing1Premium, var wing1Delta, var wing1Gamma, var wing1Theta, var wing1Vega,
var bodyPremium, var bodyDelta, var bodyGamma, var bodyTheta, var bodyVega,
var wing2Premium, var wing2Delta, var wing2Gamma, var wing2Theta, var wing2Vega,
   var quantity
);
```

7.3. BREAK-EVEN POINTS

7.3.1. BEPPOINTS Structure

Break-even point structure.

```
typedef struct {
var upper; // Upper BEP
var lower; // Lower BEP
varrange; // BEP range (upper - lower)
var rangePct; // BEP range %
var currentPrice; // Current spot
var distanceUpper; // Distance to upper
var distanceLower; // Distance to lower
  } BEPPOINTS;
```

7.3.2. calculateBEPstraddle

Calculate break-even points for STRADDLE.

```
BEPPOINTS calculateBEPStraddle(
var strike,
```

```
var netPremium,
var currentPrice
);
```

Example:

```
BEPPOINTS bep = calculateBEPStraddle(4500, 90, 4500);
printf("\nUpper BEP: $%.2f", bep.upper); // 4590
printf("\nLower BEP: $%.2f", bep.lower); // 4410
printf("\nBEP Range: $%.2f (+/-%.2f%%)", bep.range, bep.rangePct/2);
```

- **calculateBEPSpread**

Calculate break-even point for VERTICAL SPREAD.

```
BEPPOINTS calculateBEPSpread(
var shortStrike,
var longStrike,
var netCredit,
var currentPrice,
int isCallSpread // 1 = call spread, 0 = put spread
);
```

7.4. RISK/REWARD

7.4.1. calculateMaxRisk

Calculate maximum risk for spread.

```
var calculateMaxRisk(var widthSpread, var netCredit);
```

Example:

```
var width = 50; // Spread $50 wide
var credit = 15; // Credit $15
var maxRisk = calculateMaxRisk(width, credit);
printf("\nMax Risk: $%.2f", maxRisk); // $3500 = (50-15)*100
```

- **calculateStopLoss**

Calculate stop losses for credit strategies.

```
var calculateStopLoss(var maxProfit, var multiplier, var minFloor);
```

Example:

```
var maxProfit = 1500;
var stopLoss = calculateStopLoss(maxProfit, 1.5, 500);
// Stop at 150% of max profit, but minimum $500
printf("\nStop Loss: $%.2f", stopLoss);
```

7.4.2. calculateProfitTarget

Calculate profit targets for credit strategies.

var calculateProfitTarget(var maxProfit, var targetPct);

Example:

var maxProfit = 1500;

var profitTarget = calculateProfitTarget(maxProfit, 60);

// Exit at 60% of max profit

printf("\nProfit Target: $%.2f", profitTarget); // $900

7.5. POSITION ANALYSIS

7.5.1. isInsideBEP

Check if price is within BEP range.

int isInsideBEP(var currentPrice, BEPPOINTS bep);

Example:

if(isInsideBEP(priceClose(), bep)) {

printf("\nâš ï¸ Price inside BEP range - losing position");

}

7.5.2. distanceFromBEPpercent

Calculate percentage distance to the nearest BEP.

var distanceFromBEPPercent(var currentPrice, BEPPOINTS bep);

Return: Distance in % (positive = outside BEP, negative = inside)

7.6. LOGGING

7.6.1. logStrategyEntry

Professionally formatted logging for ENTRY.

void logStrategyEntry(

string strategyName,

string asset,

var strike,

var dte,

NETGREEKS ng,

BEPPOINTS bep,

var currentPrice

);

Example:

logStrategyEntry("LONG STRADDLE", "SPX", 4500, 45, ng, bep, 4500);

// Output:

// +â• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to•-

// | LONG STRADDLE ENTRY - SPX

// +â• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to• to•

//

// Date: 2025-11-10

// Price: $4500.00 Strike: $4500 DTE: 45

//

// +- POSITION METRICS --------------------------------â”

// | Net Premium: $9000.00 (DEBIT)

// | Max Profit: $9000.00

// +--â”˜

// ...

7.6.2. logStrategyExit

Professionally formatted logging for EXIT.

```
  void logStrategyExit(
string strategyName,
var finalPL,
var entryPrice,
var exitPrice,
var daysHeld,
   string exitReason
);
```

7.6.3. printGreeksSummary

Quick printing of Greeks (single line).

```
  void printGreeksSummary(NETGREEKS ng);
```

Output: Delta:0.050 Gamma:0.0012 Theta:$45.30 Vega:18.20

7.7. ANALYTICS

7.7.1. calculateImpliedMove

Calculate implied movement from straddle price.

```
  var calculateImpliedMove(var straddlePrice, var currentPrice);
```

Example:

```
var straddlePrice = 90;
var spot = 4500;
  var impliedMove = calculateImpliedMove(straddlePrice, spot);
printf("\nMarket expects +/-%.2f%% move", impliedMove); // ~2%
```

7.7.2. calculateProbabilityOTM

Estimate OTM probability from delta.

var calculateProbabilityOTM(var delta);

Example:

```
  var delta = -0.25; // Short PUT
var probOTM = calculateProbabilityOTM(delta);
printf("\n~%.0f%% probability of expiring OTM", probOTM); // 75%
```

Note: Delta approximates ITM probability, so: OTM Prob â‰ˆ 100% - |delta| Ã— 100%

7.7.3. thetaPerMinute

Converts theta daily to per-minute (for 0DTE).

```
  var thetaPerMinute(var thetaDaily);
```

Example:

```
var thetaDaily = -25; // -$25/day
var thetaMin = thetaPerMinute(thetaDaily);
printf("\nTheta: $%.4f/minute", thetaMin);
```

7.7.4. thetaPerHour

Converts theta daily to per-hour.

var thetaPerHour(var thetaDaily);

Example:

```
var thetaDaily = -25;
var thetaHour = thetaPerHour(thetaDaily);
printf("\nTheta: $%.2f/hour", thetaHour); // ~-$1.04/hour
```

8. CHAPTER 8: INPUT/OUTPUT FUNCTIONS

8.1. 📁 GENERAL INFORMATION

Input/Output functions handle all read/write, network communication, user interface, and display operations. This chapter documents critical functions for:

8.1.1. 📊 FUNCTIONS SUMMARY

This chapter includes the following key functions:

📁 File Operations (10 functions)

- Read/write: file_read, file_write, file_content, file_append
- Management: file_copy, file_delete, file_length, file_date
- Directories: file_next, file_select

🌐 **FTP Operations (10 functions)**

- Transfer: ftp_download, ftp_upload, ftp_stop
- Info: ftp_getdate, ftp_timestamp, ftp_size, ftp_sent
- Status: ftp_status, ftp_log

🌐 **HTTP (7 functions)**

- Transfer: http_transfer, http_request, http_post
- Control: http_proxy, http_free
- Status: http_status, http_result

🎨 **UI & Panels (8 features)**

- Creation: panel, panelSet, panelGet
- I/O: panelSave, panelLoad
- Layout: panelFix, panelMerge

📋 **System Variables (2 functions)**

- getvar, putvar

🔔 **Alerts (3 functions)**

- email, sound, message

📊 **Plotting : (28 functions)**

- color, colorScale, plot, plotBar, plotChart, plotData, plotGraph, plotBuyHold, plotContract, plotCorrelogram, plotDay, plotDayProfit, plotHeatmap, plotHistogram, plotMAEGraph, plotMAEPercentGraph, plotMFEGraph, plotMFEPercentGraph, plotMonth, plotMonthProfit, plotPriceProfile, plotQuarterProfit, plotTradeProfile, plotWeek, plotWeekProfit, plotWFOCycle, plotWFOProfit, plotYear

🐍 **Python Bridge: (6 functions)**

- pyStart, pyX, pySet, pyVar, pyInt, pyVec

📈 **R bridge: (7 functions)**

- Rstart, Rx, Rset, Rd, Ri, Rv, Rrun\

🖱️ **Misc UI: (8 functions)**

- printf, print, slider, progress, report, keys, mouse, window

8.1.2. General Best Practices

File Operations:

- ✅ Always check existence with file_length() before operating
- ✅ Use paths relative to Zorro/Data directory
- ✅ Backup critical files before deleting/overwriting
- ✅ Handling errors (checking return values)
- ✅ Close/reset files after use

FTP Operations:

- ✅ Implement timeouts for async operations

- ✅ Check file size after download
- ✅ Use ftp_log() for debugging, disable in prod
- ✅ Handle network errors gracefully
- ✅ Don't hardcode passwords - use config files

Performance:

- ✅ Prefer file_read() to file_content() for files > 1MB
- ✅ Batch file operations when possible
- ✅ Cache file_date() results if called frequently
- ✅ Async FTP does not block - use for long operations

Plotting:

- ✅ Use plotData() to export custom analyses
- ✅ Combine MAE/MFE to optimize stop/target
- ✅ Plot seasonality for timing entries
- ✅ Consistent colors for related series

Python/R Bridge:

- ✅ Initialize bridge in INITRUN
- ✅ Use bridge for ML, advanced stats not available in C
- ✅ Minimize data transfers (bottleneck)
- ✅ Cache results when possible

UI:

- ✅ Slider for quick manual tuning
- ✅ Progress bar for long operations
- ✅ Panel for real-time monitoring
- ✅ Automatic reports for reviews

Performance:

- ✅ Plot only necessary data
- ✅ Limit Python/R calls (overhead ~10ms)
- ✅ Use panel with update throttling
- ✅ Export large data only in EXITRUN

8.2. 📁 FILE OPERATIONS

8.2.1. file_append

Appends data to the end of an existing file. If the file doesn't exist, it is created.

Syntax

int file_append(string FileName, string Content);

Parameters

- **FileName** (string): Name or complete path of the file
- **Content** (string): Content to add

Return Value

int - Number of bytes written, 0 if error

Example of Use

```
function run()
{
// Daily trade log
string logEntry = strf("\n%s,%.2f,%.2f,%d",
strdate("%Y-%m-%d", wdate()),
ProfitTotal,
WinTotal,
NumWinTotal);

file_append("Data/daily_log.csv", logEntry);

// Export optimized parameters
if(is(OPTIMIZED)) {
string params = strf("\nSharpe=%.2f,MaxDD=%.2f", Sharpe, MaxDrawdown);
file_append("Opt_Results.txt", params);
   }
}
```

Notes

- Automatically create file if it doesn't exist
- Don't add newlines - use \n explicitly
- Useful for continuous logging without overwriting
- Warning: increasing file size
- Alternative: file_write() with mode "a"

Related Functions

- file_write() - Writes by overwriting
- file_appendfront() - Appends to the front
- printf() - Console Output

8.2.2. file_appendfront

Appends content to the BEGINNING of an existing file (prepend).

Syntax

int file_appendfront(string FileName, string Content);

Parameters

- **FileName** (string): Path of the file

- **Content** (string): Content to be inserted at the head

Return Value

int - Number of bytes written, 0 if error

Example of Use

```
function run()
{
if(is(INITRUN)) {
// Add header to existing CSV
string header = "Date,Asset,PL,MaxDD,Sharpe\n";
file_appendfront("Results/backtest.csv", header);

// Log version/timestamp in the head
string version = strf("# Version %s - %s\n",
version(), strdate("%Y-%m-%d", wdate()));
file_appendfront("strategy.log", version);
   }
}
```

Notes

- EXPENSIVE operation for large files (reads everything + rewrites)
- Prefer file_append() for frequent logging
- Useful for one-time header/metadata
- Not thread-safe for shared files

8.2.3. file_content

Reads the entire contents of a file into memory.

Syntax

```
string file_content(string FileName);
```

Parameters

- **FileName** (string): Path of the file to read

Return Value

string - Full contents of the file, NULL if error

Example of Use

```
function run()
{
// Read configuration
string config = file_content("Strategy/config.txt");
   if(config) {
```

```
// Parse parameters
var stopLoss = strvar(config, "StopLoss");
var takeProfit = strvar(config, "TakeProfit");

printf("\nConfig loaded: SL=%.1f TP=%.1f", stopLoss, takeProfit);
}

// Read custom asset list
string assetList = file_content("Data/my_assets.txt");
if(assetList) {
assetList(assetList);
}

// Read HTML template
string htmlTemplate = file_content("Templates/report.html");
  // ...customize and generate reports
}
```

Notes

- Load ENTIRE file into memory - beware of large files
- For files > 1MB use line-by-line reading
- Returns NULL if file not found
- String is zero-terminated
- Typical max size ~10MB

Best Practices

```
// Always check existence
string data = file_content("myfile.txt");
if(!data) {
printf("\nERROR: File not found!");
  return;
}
// For large files: file_read() with loop
```

Related Functions

- file_read() - Progressive reading
- file_length() - File size
- dataParse() - Parse CSV from string

8.2.4. file_copy

Copy a file from source to destination.

Syntax

int file_copy(string Source, string Destination);

Parameters

- **Source** (string): Source file path
- **Destination** (string): Path destination

Return Value

int - 1 if successful, 0 if unsuccessful

Example of Use

```
function run()
{
if(is(EXITRUN)) {
// Backup results
string timestamp = strdate("%Y%m%d_%H%M%S", wdate());
string backup = strf("Backup/results_%s.txt", timestamp);

if(file_copy("Log/strategy.log", backup)) {
printf("\nBackup created: %s", backup);
      }

// Copy best configuration
if(Sharpe > 2.0) {
         file_copy("Strategy/current.c", "Strategy/best_config.c");
}
}

// Pre-optimization backup
if(is(FIRSTRUN) && Optimize) {
file_copy("Data/AssetList.csv", "Data/AssetList_backup.csv");
   }
}
```

Notes

- Overwrite destination if it exists
- Preserve original content
- Path relative to Zorro/Data directory
- Don't create directory - path must exist
- Use \ or / for separators

Best Practices

```
// Check source existence
if(file_length("source.txt") > 0) {
file_copy("source.txt", "dest.txt");
}
// Create unique naming for backup
string backupName = strf("backup_%d.txt", frame());
```

8.2.5. file_date

Gets the file's last modification date/time.

Syntax

var file_date(string FileName);

Parameters

- **FileName** (string): Path of the file

Return Value

var - OLE DATE (serial number), 0 if file does not exist

Example of Use

```
function run()
{
// Check data freshness
var dataDate = file_date("History/SPY.t6");
   var today = wdate();
var daysOld = today - dataDate;

if(daysOld > 7) {
printf("\n!!! WARNING: Data file %d days old", (int)daysOld);
printf("\nRun AssetHistory to update");
   }

// Compare two files
var config1 = file_date("config_v1.txt");
   var config2 = file_date("config_v2.txt");

string newer = (config2 > config1) ? "v2" : "v1";
printf("\nNewer config: %s", newer);

// Check if file was modified today
if((int)file_date("trades.csv") == (int)wdate()) {
```

```
printf("\nTrades file updated today");
   }
}
```

Notes

- OLE DATE format (days since 1/1/1900)
- Accuracy: seconds
- Time zone: local system
- 0 if file does not exist
- Useful for cache invalidation

Conversions

```
var fileDate = file_date("myfile.txt");
printf("\nFile modified: %s", strdate("%Y-%m-%d %H:%M", fileDate));
```

8.2.6. file_delete

Delete a file from disk.

Syntax

```
int file_delete(string FileName);
```

Parameters

- **FileName** (string): Path file to delete

Return Value

int - 1 if successful, 0 if error or file does not exist

Example of Use

```
function run()
{
// Cleaning up temporary files
if(is(EXITRUN)) {
file_delete("temp_data.txt");
file_delete("cache.tmp");
}

// Delete old backups
   for(int i = 1; i <= 10; i++) {
string oldBackup = strf("Backup/old_%d.txt", i);
if(file_date(oldBackup) < wdate() - 30) { // > 30 days
if(file_delete(oldBackup)) {
printf("\nDeleted old backup: %s", oldBackup);
}
```

```
}
}

// Reset log if too large
if(file_length("debug.log") > 10000000) { // > 10MB
file_delete("debug.log");
printf("\nDebug log cleared");
   }
}
```

Notes

- ⚠ IRREVERSIBLE - no undo!
- Do not delete directories
- Fails if file in use/locked
- Silent if file does not exist
- Path relative to Zorro/Data

Best Practices

```
// Confirm before deleting
if(file_length("important.txt") > 0) {
   // Backup before delete
file_copy("important.txt", "important_backup.txt");
file_delete("important.txt");
}

// Delete only if certain
if(is(EXITRUN) && TestMode) {
file_delete("temp.txt");
}
```

8.2.7. file_length

Returns file size in bytes.

Syntax

long file_length(string FileName);

Parameters

- **FileName** (string): Path of the file

Return Value

long - Size in bytes, 0 if file does not exist or is empty

Example of Use

```
function run()
{
// Check data availability
long historySize = file_length("History/SPY.t6");
if(historySize == 0) {
printf("\nERROR: History file missing!");
quit("Download price data first");
}

printf("\nHistory size: %.2f MB", historySize / 1048576.0);

// Limit log size
if(file_length("trades.log") > 5000000) { // 5MB
// Backup and reset
file_copy("trades.log", "trades_old.log");
      file_delete("trades.log");
}

// Check download completeness
   long expectedSize = 1000000; // 1MB
long actualSize = file_length("downloaded_data.csv");

if(actualSize < expectedSize) {
printf("\n!!! Incomplete download: %ld/%ld bytes",
           actualSize, expectedSize);
}
}
```

Notes

- Returns 0 even for empty files
- It does not distinguish "does not exist" from "size 0"
- Max file size: 2^63-1 bytes
- Fast - does not read content

Useful Conversions

```
long bytes = file_length("file.bin");
printf("\nSize: %.2f KB", bytes / 1024.0);
printf("\nSize: %.2f MB", bytes / 1048576.0);
printf("\nSize: %.2f GB", bytes / 1073741824.0);
```

8.2.8. file_next

Iterates through files in a directory (glob pattern).

Syntax

string file_next(string Pattern);

Parameters

- **Pattern** (string): Glob pattern (e.g. " *.csv", "backup_* .txt")

Return Value

string - Next file name, NULL when list is exhausted

Example of Use

```
function run()
{
if(is(INITRUN)) {
// List all CSVs in the directory
printf("\n=== CSV Files Found ===");
string file = file_next("Data/*.csv");
int count = 0;

while(file) {
printf("\n%d. %s (%.2f KB)",
++count, file, file_length(file)/1024.0);
        file = file_next(NULL); // NULL = continue iteration
      }

printf("\nTotal files: %d", count);
}

// Process all backups
string backup = file_next("Backup/strategy_*.c");
while(backup) {
// Analyze each backup
var backupDate = file_date(backup);
printf("\nBackup: %s from %s",
backup, strdate("%Y-%m-%d", backupDate));

      backup = file_next(NULL);
}
}
```

Notes

- First call: complete pattern
- Next: file_next(NULL) continues
- Returns ONLY the file name (not the path)
- Non-recursive (specified directory only)
- Wildcards: * (multi) ? (single)
- Reset: Call with new pattern

Valid Patterns

file_next("*.t6") // All .t6

file_next("SPY_*.csv") // SPY_something.csv

file_next("data_202?.txt") // data_2020.txt, data_2021.txt, etc.

8.2.9. file_read

Read file contents progressively with buffer.

Syntax

int file_read(string FileName, string Buffer, int Count);

Parameters

- **FileName** (string): Path of the file
- **Buffer** (string): Buffer where to copy the read data
- **Count** (int): Number of bytes to read (max 4096)

Return Value

int - Number of bytes actually read, 0 at end of file

Example of Use

```
function run()
{
// Progressive reading of large file
string buffer = "";
   int bytesRead = 0;
int totalLines = 0;

do {
bytesRead = file_read("LargeFile.csv", buffer, 4096);

if(bytesRead > 0) {
// Count lines in the chunk
for(int i = 0; i < strlen(buffer); i++) {
if(strxc(buffer, i) == '\n') totalLines++;
}
```

```
}
} while(bytesRead > 0);

printf("\nTotal lines: %d", totalLines);

   // Parse CSV progressively
string line = "";
   file_read("data.csv", line, 4096);

// Split by newlines
string token = strtok(line, "\n");
while(token) {
// Process single row
printf("\nLine: %s", token);
      token = strtok(NULL, "\n");
   }
}
```

Notes

- Max 4096 bytes per read
- File remains "open" between subsequent calls
- Reset with new FileName
- More efficient than file_content() for large files
- Buffer must be pre-allocated

Best Practices

```
// Safe reading with loop
int total = 0;
string chunk = "";
int read;

while((read = file_read("big.dat", chunk, 4096)) > 0) {
total += read;
// Process chunk
}

printf("\nTotal bytes read: %d", total);
```

8.2.10. file_select

Show file selection dialog (GUI).

Syntax

string file_select(string Caption, string Filter);

Parameters

- **Caption** (string): Dialog window title
- **Filter** (string): Extension filter (e.g., " *.csv;* .txt")

Return Value

string - Full path to the selected file, NULL if canceled

Example of Use

```
function run()
{
if(is(INITRUN)) {
// Select configuration file
	string configFile = file_select(
"Select Strategy Config",
"*.ini;*.cfg;*.txt"
);

if(configFile) {
printf("\nLoading config: %s", configFile);
string config = file_content(configFile);
// Parse config...
} else {
printf("\nNo config selected, using defaults");
}

// Select custom dataset
string dataFile = file_select(
"Choose Price Data",
"*.csv;*.t6"
);

if(dataFile) {
// Load custom data
		assetHistory(dataFile);
}
}
}
```

Notes

- ⚠ BLOCK execution until selected!
- GUI mode only (not headless/server)
- Return path ABSOLUTE
- NULL if user deletes
- Filter Windows syntax: "*.ext1;*.ext2"

Filter Examples

"*.*" // All files

"*.csv" // CSV only

"*.c;*.h" // C source and headers

"Data*.txt" // Data... .txt

8.2.11. file_write

Writes content to file (overwrite or append).

Syntax

int file_write(string FileName, string Content, int Mode);

Parameters

- **FileName** (string): Path to destination file
- **Content** (string): Data to write
- **Mode** (int): 0=overwrite, 1=append

Return Value

int - Number of bytes written, 0 if error

Example of Use

```
function run()
{
// Overwrite file
string report = strf("Sharpe: %.2f\nMaxDD: %.2f%%\n",
Sharpe, MaxDrawdown);
file_write("Report.txt", report, 0);

// Append to existing file
string logEntry = strf("%s,%.2f,%.2f\n",
strdate("%Y-%m-%d", wdate()),
WinTotal, LossTotal);
file_write("daily_log.csv", logEntry, 1);

  // Save optimized configuration
```

```
if(is(OPTIMIZED)) {
	string config = strf(
"# Optimal Parameters\n"
"Period = %d\n"
"StopLoss = %.1f\n"
"TakeProfit = %.1f\n",
optimize(20, 10, 100, 10), // example
optimize(20, 10, 50, 5),
optimize(40, 20, 100, 10)
);

file_write("Config/optimal.ini", config, 0);
  }
}
```

Notes

- Mode 0: erase and rewrite (default)
- Mode 1: append (like file_append())
- Create file if it does not exist
- No automatic newline - add \n
- Path relative to Zorro/Data

Best Practices

```
// CSV export
string csv = "Date,PL,Equity\n";
for(int i = 0; i < 100; i++) {
csv = strcat(csv, strf("%d,%.2f,%.2f\n", i, PL[i], Equity[i]));
}
file_write("export.csv", csv, 0);

// Structured logging
file_write("log.txt", strf("[%s] %s\n",
strdate("%H:%M:%S", wdate()),
  "Strategy started"
), 1); // append
```

8.3. FTP OPERATIONS

8.3.1. ftp_download

Download files from FTP servers.

Syntax

```
int ftp_download(
string RemoteFile,
string LocalFile,
string Server,
string User,
Password string
);
```

Parameters

- **RemoteFile** (string): Path file on FTP server
- **LocalFile** (string): Local destination path
- **Server** (string): FTP server address (e.g. "ftp.example.com")
- **User** (string): FTP Username
- **Password** (string): FTP password

Return Value

int - 1 during download, 2 when complete, 0 if error

Example of Use

```
function run()
{
static int downloadStatus = 0;

if(is(INITRUN)) {
// Start downloading EOD data
downloadStatus = ftp_download(
"/data/prices/SPY_2024.csv",
"History/SPY_2024.csv",
"ftp.datavendor.com",
"myuser",
"mypass"
);

printf("\nDownload started...");
}

// Check the status of each bar
if(downloadStatus == 1) {
downloadStatus = ftp_download(NULL,NULL,NULL,NULL,NULL);
```

```
if(downloadStatus == 2) {
printf("\nDownload completed!");
long size = ftp_size();
printf("\nReceived: %.2f KB", size/1024.0);

        // Load downloaded data
assetHistory("SPY_2024.csv");
}
}
}
```

Notes

- **ASYNCHRONOUS** - does not block execution
- First call: start download (return 1)
- Subsequent calls with NULL: check status
- Status 1 = in progress, 2 = done, 0 = error
- Default timeout: 60 seconds
- Supports FTP/FTPS (not SFTP)

Best Practices

```
// Complete pattern with timeout
static int ftpState = 0;
static int ftpTimeout = 0;

if(is(INITRUN)) {
ftpState = ftp_download(...);
ftpTimeout = 60; // 60 bars timeout
}

if(ftpState == 1) {
ftpState = ftp_download(NULL,NULL,NULL,NULL,NULL);

if(--ftpTimeout <= 0) {
ftp_stop();
printf("\n!!! FTP Timeout");
}
}
```

8.3.2. ftp_upload

Upload files to FTP server.

Syntax

```
int ftp_upload(
string LocalFile,
string RemoteFile,
string Server,
string User,
Password string
);
```

Parameters

- **LocalFile** (string): Local file to upload
- **RemoteFile** (string): Path destination on server
- **Server** (string): FTP server address
- **User** (string): FTP username
- **Password** (string): FTP password

Return Value

int - 1 during upload, 2 when completed, 0 if error

Example of Use

```
function run()
{
static int uploadStatus = 0;

if(is(EXITRUN)) {
	// Load daily report
string date = strdate("%Y%m%d", wdate());
	string localReport = strf("Reports/daily_%s.html", date);
string remoteReport = strf("/public_html/reports/daily_%s.html", date);

uploadStatus = ftp_upload(
localReport,
remoteReport,
"ftp.mysite.com",
"webuser",
"webpass"
);

printf("\nUploading report...");
```

```
}

// Monitor upload progress
if(uploadStatus == 1) {
uploadStatus = ftp_upload(NULL,NULL,NULL,NULL,NULL);

if(uploadStatus == 2) {
printf("\nReport uploaded successfully!");
printf("\nSent: %.2f KB", ftp_sent()/1024.0);
}
}

// Backup strategy on remote server
if(Optimize && is(OPTIMIZED)) {
ftp_upload(
"Strategy/optimal.c",
"/backups/optimal_latest.c",
"backup.server.com",
"backup_user",
        "backup_pass"
);
}
}
```

Notes

- ASYNCHRONOUS like ftp_download()
- Create remote directories if they don't exist (YMMV)
- Overwrite existing files
- Max file size: limited by server
- Progress tracking via ftp_sent()

Error Handling

```
if(uploadStatus == 0) {
printf("\n!!! FTP Upload failed");
printf("\nCheck: network, credentials, permissions");
}
```

8.3.3. ftp_getdate

Gets file timestamps on FTP server.

Syntax

```
var ftp_getdate(
string RemoteFile,
string Server,
string User,
   Password string
);
```

Parameters

- **RemoteFile** (string): Remote file path
- **Server** (string): FTP server
- **User** (string): Username
- **Password** (string): Password

Return Value

var - OLE DATE of the file, 0 if error

Example of Use

```
function run()
{
if(is(INITRUN)) {
// Check remote data freshness
var remoteDate = ftp_getdate(
"/data/SPY.csv",
"ftp.vendor.com",
"user",
"pass"
);

var localDate = file_date("History/SPY.csv");

if(remoteDate > localDate) {
printf("\nRemote data newer - downloading...");
ftp_download(...);
} else {
printf("\nLocal data up to date");
      }

// Check last update
var daysSince = wdate() - remoteDate;
      if(daysSince > 1) {
```

```
printf("\n!!! WARNING: Data %d days old", (int)daysSince);
    }
}
}
```

Notes

- **SYNCHRONOUS** - block until answered
- Timeout: ~10 seconds
- 0 if file does not exist or error
- Timezone: Local FTP Server
- Useful for cache validation

8.3.4. ftp_stop

Stop FTP transfer in progress.

Syntax

```
void ftp_stop();
```

Parameters

Nobody

Return Value

Void

Example of Use

```
function run()
{
static int ftpStatus = 0;
static int timeout = 0;

if(is(INITRUN)) {
ftpStatus = ftp_download(...);
timeout = 120; // 2 minutes max
}

if(ftpStatus == 1) {
ftpStatus = ftp_download(NULL,NULL,NULL,NULL,NULL);

// Timeout protection
if(--timeout <= 0) {
ftp_stop();
printf("\n!!! Download timeout - aborted");
```

```
ftpStatus = 0;
}

// User abort
if(key(VK_ESCAPE)) {
ftp_stop();
printf("\nDownload canceled by user");
	ftpStatus = 0;
}
}
}
```

Notes

- Stop IMMEDIATELY
- Partial file may remain
- Clear FTP connection
- Safe to call even if no transfer is active
- Does not reset status code - check manually

8.3.5. ftp_size

Gets the size of the file received via FTP.

Syntax

long ftp_size();

Parameters

Nobody

Return Value

long - Bytes received during last download, 0 if none

Example of Use

```
function run()
{
static int dlStatus = 0;

if(is(INITRUN)) {
dlStatus = ftp_download(...);
}

if(dlStatus == 1) {
dlStatus = ftp_download(NULL,NULL,NULL,NULL,NULL);
```

```
// Progress indicator
long received = ftp_size();
printf("\rDownloaded: %.2f KB", received/1024.0);

if(dlStatus == 2) {
printf("\nDownload complete: %.2f MB", received/1048576.0);

// Check integrity
long fileSize = file_length("downloaded.dat");
if(fileSize != received) {
printf("\n!!! Size mismatch: expected %ld, got %ld",
                received, fileSize);
}
}
}
}
```

Notes

- Updated in real time during download
- Reset to 0 for new download
- Includes header/metadata (not just payload)
- Useful for progress bars
- Not available for upload (use ftp_sent())

8.3.6. ftp_sent

Gets the size of data sent via FTP upload.

Syntax

```
long ftp_sent();
```

Parameters

Nobody

Return Value

long - Bytes sent during last upload, 0 if none

Example of Use

```
function run()
{
static int upStatus = 0;
static long totalSize = 0;
```

```
if(is(EXITRUN)) {
totalSize = file_length("Report.html");
upStatus = ftp_upload(...);
printf("\nUploading %.2f KB...", totalSize/1024.0);
}

if(upStatus == 1) {
upStatus = ftp_upload(NULL,NULL,NULL,NULL,NULL);

// Progress bar
long sent = ftp_sent();
int percent = (int)((sent * 100) / totalSize);
printf("\rProgress: %d%% (%.1f/%.1f KB)",
percent, sent/1024.0, totalSize/1024.0);

if(upStatus == 2) {
        printf("\nUpload complete!");
}
}
}
```

Notes

- Counterpart of ftp_size() for upload
- Updated in real time
- Includes FTP protocol overhead
- Reset for new upload

8.3.7. ftp_timestamp

Get file date/time from FTP server (precise timestamp).

Syntax

```
long ftp_timestamp(
string RemoteFile,
string Server,
string User,
Password string
);
```

Parameters

- **RemoteFile** (string): Remote file path
- **Server** , **User** , **Password** : FTP Credentials

Return Value

long - Unix timestamp (seconds since 1970), 0 on error

Example of Use

```
function run()
{
// Precise timestamp for sync
long remoteTime = ftp_timestamp(
"/data/prices_latest.csv",
"ftp.vendor.com",
"user",
"pass"
);

if(remoteTime > 0) {
// Convert to OLE date
var oleDate = mtu(remoteTime);
printf("\nRemote file timestamp: %s",
strdate("%Y-%m-%d %H:%M:%S", oleDate));

// Compare with local
long localTime = utm(file_date("prices_latest.csv"));

if(remoteTime > localTime) {
printf("\nRemote version newer by %ld seconds",
        remoteTime - localTime);
}
}
}
```

Notes

- More precise than ftp_getdate() (seconds vs days)
- SYNCHRONOUS - may delay
- Unix timestamp (epoch time)
- Timezone: UTC typically
- Convert with mtu() for OLE dates

Conversions

```
long unix_time = ftp_timestamp(...);
```

```
var ole_date = mtu(unix_time); // Unix -> OLE
long back_to_unix = utm(ole_date); // OLE -> Unix
```

8.3.8. ftp_status

Gets the current status of the FTP operation.

Syntax

```
int ftp_status();
```

Parameters

Nobody

Return Value

int - Status code: 0=idle/error, 1=in progress, 2=completed

Example of Use

```
function run()
{
static int started = 0;

if(is(INITRUN) && !started) {
ftp_download(...);
started = 1;
}

// Poll status each bar
int status = ftp_status();

switch(status) {
case 0:
if(started) {
printf("\n!!! FTP Error or not started");
}
break;

case 1:
printf("\rDownloading... %.1f KB", ftp_size()/1024.0);
break;

case 2:
printf("\nDownload finished!");
```

```
started = 0; // Reset for next
break;
}

// Error handling
if(status == 0 && started && ftp_size() == 0) {
printf("\nPossible causes:");
printf("\n- Network down");
printf("\n- Wrong credentials");
printf("\n- File not found");
   }
}
```

Notes

- Alternative to passing NULL to ftp_download/upload()
- Does not provide specific error details
- Safe to call anytime
- Automatic reset after completion

8.3.9. ftp_log

Enable detailed logging of FTP operations.

Syntax

void ftp_log(int Enable);

Parameters

- **Enable** (int): 1=enable logging, 0=disable

Return Value

Void

Example of Use

```
function run()
{
if(is(INITRUN)) {
// Enable FTP debugging
ftp_log(1);

ftp_download(
"/data/test.csv",
"test.csv",
"ftp.server.com",
```

```
"user",
"pass"
);
}
}

// Output to Zorro log:
// FTP: Connecting to ftp.server.com...
// FTP: Connected
// FTP: USER user
// FTP: PASS ****
// FTP: TYPE I
// FTP: PASV
// FTP: RETR /data/test.csv
// FTP: 1024 bytes received
// FTP: 2048 bytes received
// FTP: Transfer complete: 2048 bytes
// FTP: QUIT
```

Notes

- Log written in Zorro main log
- Useful for debugging connections
- Show FTP commands and responses
- Masked password (****)
- Minimal overhead when disabled
- Disable in production for performance

Best Practices

```
// Debug mode
if(DebugMode) {
ftp_log(1);
}

// Production
ftp_log(0);
```

8.4. 🌐 HTTP OPERATIONS

8.4.1. http_transfer

Download content from HTTP/HTTPS URLs.

Syntax

int http_transfer(string URL, string LocalFile);

Parameters

- **URL** (string): Full resource URL (http:// or https://)
- **LocalFile** (string): Local destination path (NULL for memory)

Return Value

int - 1 during download, 2 completed, 0 error, -1 timeout

Example of Use

```
function run()
{
static int httpStatus = 0;

if(is(INITRUN)) {
    // Download EOD prices from API
httpStatus = http_transfer(
"https://api.datavendor.com/v1/prices/SPY?format=csv",
        "History/SPY_latest.csv"
);

printf("\nDownloading market data...");
}

  // Poll status
if(httpStatus == 1) {
httpStatus = http_transfer(NULL, NULL);

if(httpStatus == 2) {
printf("\nDownload complete!");

        // Load data
string data = http_result();
printf("\nReceived %d bytes", strlen(data));

        // Parse CSV
dataParse(data, ", ");
}
else if(httpStatus == -1) {
```

```
        printf("\n!!! HTTP Timeout");
}
}

  // Download configuration from GitHub
http_transfer(
"https://raw.githubusercontent.com/user/repo/main/config.ini",
    "Config/remote_config.ini"
);
}
```

Notes

- **ASYNCHRONOUS** - does not block
- Supports HTTP and HTTPS
- Default timeout: 60 seconds
- LocalFile=NULL: save to memory (use http_result())
- Subsequent calls with NULL: check status
- Automatic headers (User-Agent, Accept, etc.)

Status Codes

- 1 = Transfer in progress
- 2= Transfer completed successfully
- 0 = Error (network, DNS, 404, etc.)
- -1 = Timeout

8.4.2. http_request

Send custom HTTP request with headers/body.

Syntax

int http_request(string URL, string Headers, string Body);

Parameters

- **URL** (string): Endpoint URL
- **Headers** (string): Custom HTTP headers (NULL by default)
- **Body** (string): Request body for POST/PUT (NULL for GET)

Return Value

int - HTTP status code (200, 404, 500, etc.), 0 if error

Example of Use

```
function run()
{
  // REST API call with authentication
```

```
string headers = "Authorization: Bearer YOUR_API_KEY\n"
"Content-Type: application/json\n";

string body = "{"
"\"symbol\": \"SPY\","
"\"interval\": \"1d\","
"\"range\": \"1mo\""
"}";

int status = http_request(
"https://api.broker.com/v2/marketdata",
headers,
body
);

if(status == 200) {
string response = http_result();
      printf("\nAPI Response: %s", response);

      // Parse JSON response
      // ... (use dataParse or manual parsing)
   }
else {
printf("\nHTTP Error: %d", status);
}

   // Webhook notification
string webhook = "{"
"\"text\": \"Trade executed: BUY SPY @450\","
"\"channel\": \"#trading-alerts\""
"}";

http_request(
"https://hooks.slack.com/services/YOUR/WEBHOOK/URL",
"Content-Type: application/json\n",
      webhook
);
}
```

Notes

- **SYNCHRONOUS** - block until answered
- Timeout: ~30 seconds
- Headers separated by \n
- Body=NULL → GET request
- Body present → POST request
- Response in http_result()

Common Status Codes

- 200 = OK
- 201 = Created
- 400 = Bad Request
- 401 = Unauthorized
- 404 = Not Found
- 429 = Too Many Requests (rate limit)
- 500 = Server Error

8.4.3. http_post

Send data via HTTP POST (form data).

Syntax

int http_post(string URL, string Data);

Parameters

- **URL** (string): Endpoint URL
- **Data** (string): Form data (format: "key1=value1&key2=value2")

Return Value

int - HTTP status code, 0 if error

Example of Use

```
function run()
{
if(is(EXITRUN)) {
    // Send trading results to dashboard
string formData = strf(
"sharpe=%.2f&"
"max_dd=%.2f&"
"total_pl=%.2f&"
"trades=%d",
Sharpe,
MaxDrawdown,
ProfitTotal,
```

```
NumTrades
);

int status = http_post(
"https://dashboard.mysite.com/api/submit",
formData
);

if(status == 200) {
printf("\nResults submitted successfully");
}

    // Login to external service
string credentials = "username=trader&password=secret123";

status = http_post(
"https://api.service.com/auth/login",
credentials
);

if(status == 200) {
string token = http_result();
printf("\nLogged in, token: %s", token);
    }
}
}
```

Notes

- SYNCHRONOUS - block
- Automatic Content-Type: application/x-www-form-urlencoded
- Automatic URL encoding for values
- Response in http_result()
- Max data size: ~64KB

Form Encoding

// Automatically encode spaces and special characters
"name=John Doe&age=30" → "name=John%20Doe&age=30"

8.4.4. http_proxy

Configure proxy server for HTTP/HTTPS requests.

Syntax

void http_proxy(string ProxyServer);

Parameters

- **ProxyServer** (string): Proxy address (format: "host:port")

Return Value

Void

Example of Use

```
function run()
{
if(is(INITRUN)) {
    // Configure corporate proxy
http_proxy("proxy.company.com:8080");

    // Authenticated proxy (if supported)
http_proxy("user:pass@proxy.company.com:8080");

    // Disable proxy
http_proxy(NULL);

    // Now all http_* will use the proxy
    http_transfer("https://api.example.com/data", "data.json");
  }
}
```

Notes

- Global for all HTTP requests
- Format: [user:pass@]host:port
- NULL disable proxy
- Does not support SOCKS proxy (HTTP/HTTPS only)
- Check company firewall/permissions

8.4.5. http_status

Gets the current HTTP transfer status.

Syntax

int http_status();

Parameters

Nobody

Return Value

int - Status: 0=idle, 1=in progress, 2=completed, -1=error

Example of Use

```
function run()
{
static int transfer = 0;
static int timeout = 0;

if(is(INITRUN)) {
transfer = http_transfer("https://api.data.com/large.csv", "data.csv");
timeout = 120; // 2 min timeout
}

  // Monitor transfer
int status = http_status();

if(status == 1) {
printf("\rDownloading... ");

if(--timeout <= 0) {
http_free();
printf("\n!!! Timeout");
}
}
else if(status == 2) {
printf("\nDownload complete!");
}
else if(status == -1) {
printf("\n!!! Transfer failed");
}
}
```

8.4.6. http_result

Gets HTTP response content.

Syntax

string http_result();

Parameters

Nobody

Return Value

string - Response content, NULL if empty/error

Example of Use

```
function run()
{
   // Synchronous API call
int status = http_request(
"https://api.forex.com/rates/EURUSD",
NULL,
NULL
);

if(status == 200) {
string json = http_result();
printf("\nJSON Response: %s", json);

      // Manual parse (simple example)
      // {"rate": 1.0850, "timestamp": 1234567890}
       string rateStr = strstr(json, "\"rate\":");
var rate = atof(rateStr + 7); // Skip "rates":

printf("\nEURUSD Rate: %.4f", rate);
}

   // Download HTML page
http_transfer("https://news.site.com/market", NULL);
if(http_status() == 2) {
string html = http_result();

      // Extract title
string titleStart = strstr(html, "<title>");
string titleEnd = strstr(titleStart, "</title>");
      // ... parsing HTML
}
}
```

Notes

- Available after status=2 (completed)

- NULL if no transfer or error
- Max size: ~10MB typically
- Dynamically allocated string
- Valid until next HTTP request

8.4.7. http_free

Stops an ongoing HTTP transfer and frees resources.

Syntax

void http_free();

Parameters

Nobody

Return Value

Void

Example of Use

c

```
function run()
{
static int dlStatus = 0;
static int timeout = 0;

if(is(INITRUN)) {
dlStatus = http_transfer("https://large-file.com/data.zip", "data.zip");
timeout = 60;
}

if(dlStatus == 1) {
dlStatus = http_transfer(NULL, NULL);

    // Timeout management
if(--timeout <= 0) {
http_free();
printf("\n!!! Download aborted - timeout");
dlStatus = 0;
}

    // User cancel
if(key(VK_ESCAPE)) {
```

```
http_free();
printf("\nDownload cancelled");
dlStatus = 0;
}
}

  // Cleanup on exit
if(is(EXITRUN)) {
http_free();
  }
}
```

Notes

- Immediate termination
- Partial file may remain
- Free up memory/connections
- Safe calling even if no transfer
- Self-called between runs (usually)

8.5. 🎨 UI & PANEL OPERATIONS

8.5.1. panel

Create custom UI panel from CSV/spreadsheet file.

Syntax

int panel(int ID, int Rows, int Cols, int W, int H, int Flags);

Parameters

- **ID** (int): Unique Panel ID (1-100)
- **Rows** (int): Number of rows
- **Cols** (int): Number of columns
- **W** (int): Cell width in pixels
- **H** (int): Cell height in pixels
- **Flags** (int): Optional Flags (PANEL_FLAG_*)

Return Value

int - Handle panel, 0 if error

Example of Use

```
function run()
{
if(is(INITRUN)) {
```

```
	// Create control panel
panel(1, 10, 5, 120, 25, 0);

	// Setup header
panelSet(1, 0, 0, "Parameter", 0, COLOR_YELLOW);
panelSet(1, 0, 1, "Value", 0, COLOR_YELLOW);
panelSet(1, 0, 2, "Min", 0, COLOR_YELLOW);
panelSet(1, 0, 3, "Max", 0, COLOR_YELLOW);
panelSet(1, 0, 4, "Optimal", 0, COLOR_YELLOW);

	// Strategy parameters
panelSet(1, 1, 0, "Period", 0, 0);
panelSet(1, 1, 1, strf("%d", Period), 0, COLOR_GREEN);
panelSet(1, 1, 2, "10", 0, 0);
panelSet(1, 1, 3, "100", 0, 0);
panelSet(1, 1, 4, "42", 0, COLOR_LIGHTBLUE);

panelSet(1, 2, 0, "StopLoss", 0, 0);
panelSet(1, 2, 1, strf("%.1f", StopLoss), 0, COLOR_RED);
}

  // Real-time updates
if(frame() % 10 == 0) {
panelSet(1, 5, 1, strf("%.2f", ProfitTotal), 0,
ProfitTotal > 0 ? COLOR_GREEN : COLOR_RED);
  }
}
```

Notes

- Create customizable grid tables
- Max 100 simultaneous panels
- Cells support text, numbers, colors
- Events: click via click() callback
- Persistent between runs

Available Flags

0 = Default (no special flags)

PANEL_FIXED = Not scrollable

PANEL_MODAL = Modal dialog

8.5.2. panelSet

Set panel cell contents/properties.

Syntax

int panelSet(int ID, int Row, int Col, string Text, var Value, int Color);

Parameters

- **ID** (int): Panel ID
- **Row** (int): Row (0-based)
- **Col** (int): Column (0-based)
- **Text** (string): Text to display (NULL for numeric value only)
- **Value** (var): Numeric value (0 if text only)
- **Color** (int): RGB color (0 by default)

Return Value

int - 1 if successful, 0 if unsuccessful

Example of Use

```
function run()
{
  // Plain text
panelSet(1, 0, 0, "Strategy Name", 0, COLOR_WHITE);

  // Numeric value
panelSet(1, 1, 1, NULL, Sharpe, COLOR_GREEN);

  // Text + conditional color
int color = WinTotal > LossTotal ? COLOR_GREEN : COLOR_RED;
panelSet(1, 2, 0, strf("P/L: $%.2f", WinTotal - LossTotal), 0, color);

  // Custom format
panelSet(1, 3, 1, strf("%.2f%%", MaxDrawdown * 100), 0, COLOR_YELLOW);

  // ASCII progress bar
int progress = (int)((NumTrades * 50) / MaxTrades);
string bar = "[";
for(int i = 0; i < 50; i++) {
bar = strcat(bar, i < progress ? "=" : " ");
}
bar = strcat(bar, "]");
panelSet(1, 4, 0, bar, 0, COLOR_CYAN);
}
```

Notes

- Text=NULL: show only Value
- Value=0: show only Text
- Color=0: use default theme
- Update visuals immediately
- Limited Unicode support

Default Colors

COLOR_BLACK, COLOR_WHITE
COLOR_RED, COLOR_GREEN, COLOR_BLUE
COLOR_YELLOW, COLOR_CYAN, COLOR_MAGENTA
COLOR_LIGHTBLUE, COLOR_LIGHTGRAY, COLOR_DARKGRAY

8.5.3. panelGet

Read panel cell contents.

Syntax

var panelGet(int ID, int Row, int Col);

Parameters

- **ID** , **Row** , **Col** : Cell identifiers

Return Value

var - Numeric value of the cell, 0 if text only

Example of Use

```
function click(int ID, int Row, int Col)
{
  // User clicked cell
if(ID == 1) {
var value = panelGet(1, Row, Col);
printf("\nClicked [%d,%d] = %.2f", Row, Col, value);

    // Toggle on click
if(Row == 5 && Col == 0) {
var current = panelGet(1, 5, 0);
panelSet(1, 5, 0, NULL, !current,
current ? COLOR_RED : COLOR_GREEN);
    }
}
}
```

8.5.4. panelSave

Save current panel state to file.

Syntax

int panelSave(int ID, string FileName);

Parameters

- **ID** (int): Panel ID
- **FileName** (string): Path file CSV output

Return Value

int - 1 if successful, 0 if unsuccessful

Example of Use

```
function run()
{
   if(is(EXITRUN)) {
      // Export results panel
panelSave(1, "Reports/panel_state.csv");

      // Configuration backup
string timestamp = strdate("%Y%m%d_%H%M%S", wdate());
string backup = strf("Backup/panel_%s.csv", timestamp);
      panelSave(1, backup);
}
}
```

8.5.5. panelLoad

Load panel status from CSV file.

Syntax

int panelLoad(int ID, string FileName);

Parameters

- **ID** (int): Panel ID
- **FileName** (string): Path CSV file

Return Value

int - 1 if successful, 0 if unsuccessful

Example of Use

```
function run()
{
if(is(INITRUN)) {
      // Restore saved state
```

```
if(file_length("panel_state.csv") > 0) {
panelLoad(1, "panel_state.csv");
printf("\nPanel state restored");
    }
}
}
```

8.5.6. panelFix

Defines the scrollable panel area.

Syntax

void panelFix(int ID, int FixRows, int FixCols);

Parameters

- **ID** (int): Panel ID
- **FixRows** (int): Fixed number of rows (header)
- **FixCols** (int): Number of fixed columns (left sidebar)

Return Value

Void

Example of Use

```
function run()
{
if(is(INITRUN)) {
    // Large panel with fixed header
panel(1, 100, 10, 100, 20, 0);

    // Header row remains visible during scroll
     panelFix(1, 1, 0);

    // Populate header
panelSet(1, 0, 0, "Symbol", 0, COLOR_YELLOW);
panelSet(1, 0, 1, "Price", 0, COLOR_YELLOW);
    // ...

    // Data rows
for(int i = 1; i < 100; i++) {
panelSet(1, i, 0, strf("SYM%d", i), 0, 0);
      // ...
}
```

```
}
}
```

8.5.7. panelMerge

Merge adjacent cells into panels.

Syntax

int panelMerge(int ID, int Row, int Col, int Rows, int Cols);

Parameters

- **ID** (int): Panel ID
- **Row** , **Col** (int): Top-left cell
- **Rows** , **Cols** (int): Number of cells to merge

Return Value

int - 1 if successful, 0 if unsuccessful

Example of Use

```
function run()
{
if(is(INITRUN)) {
panel(1, 10, 5, 100, 25, 0);

    // Title spanning integer width
panelMerge(1, 0, 0, 1, 5);
panelSet(1, 0, 0, "STRATEGY CONTROL PANEL", 0, COLOR_LIGHTBLUE);

    // Section headers
panelMerge(1, 2, 0, 1, 3);
panelSet(1, 2, 0, "--- Parameters ---", 0, COLOR_YELLOW);

panelMerge(1, 6, 0, 1, 3);
panelSet(1, 6, 0, "--- Statistics ---", 0, COLOR_GREEN);
  }
}
```

8.6. 📋 SYSTEM VARIABLES

8.6.1. getvar

Global system variable law.

Syntax

var getvar(string VarName);

Parameters

- **VarName** (string): Variable name

Return Value

var - Variable value, 0 if it does not exist

Example of Use

```
function run()
{
  // Communication between strategies
var signal = getvar("GlobalSignal");

  if(signal > 0) {
printf("\nReceived signal: %.0f", signal);
enterLong();
}

  // Shared counter
var runCount = getvar("TotalRuns");
printf("\nTotal runs across all instances: %.0f", runCount);

  // Global Configuration
var debugMode = getvar("DebugEnabled");
if(debugMode) {
    // Extra logging
}
}
```

Notes

- Shared between ALL Zorro instances
- Persistent during session
- Reset a Zorro restart
- Case-sensitive
- Max ~1000 variables

8.6.2. putvar

Write system global variable.

Syntax

void putvar(string VarName, var Value);

Parameters

- **VarName** (string): Variable name
- **Value** (var): Value to set

Return Value

Void

Example of Use

```
function run()
{
  // Broadcast signal
if(crossOver(SMA(Close, 20), SMA(Close, 50))) {
putvar("GlobalSignal", 1); // BUY
}
else if(crossUnder(SMA(Close, 20), SMA(Close, 50))) {
putvar("GlobalSignal", -1); // SELL
}

  // Increment shared counter
var count = getvar("TradeCount");
putvar("TradeCount", count + 1);

  // Share optimization result
if(is(OPTIMIZED)) {
putvar("BestSharpe", Sharpe);
putvar("BestPeriod", Period);
  }
}
```

Notes

- Create variable if it does not exist
- Overwrite if exists
- Thread-safe (atomic)
- Useful for multi-strategy coordination

8.7. 🔔 ALERT & NOTIFICATION

8.7.1. e-mail

Send email with SMTP.

Syntax

int email(string To, string Subject, string Body, string Attachment);

Parameters

- **To** (string): Email recipient
- **Subject** (string): Object
- **Body** (string): Message body
- **Attachment** (string): Path to the attachment file (NULL if none)

Return Value

int - 1 if sent, 0 if error

Example of Use

```
function run()
{
  // Critical drawdown alert
if(MaxDrawdown > 0.15) {
string body = strf(
"ALERT: Max Drawdown exceeded 15%%!\n\n"
"Current DD: %.2f%%\n"
"Strategy: %s\n"
"Date: %s",
MaxDrawdown * 100,
Script,
strdate("%Y-%m-%d %H:%M", wdate())
);

e-mail(
"trader@example.com",
"[ALERT] High Drawdown",
body,
NULL
);
}

  // Daily report with attachment
if(is(EXITRUN) && dow() == 5) { // Friday
report(); // Generate PDF

e-mail(
"boss@company.com",
"Weekly Trading Report",
```

```
"See attached performance report.",
"Report.pdf"
);
}

  // Trade notification
if(NumOpenLong > NumOpenLong[1]) {
e-mail(
"mobile@sms-gateway.com",
"Trade",
strf("BUY %s @ %.2f", Asset, price()),
        NULL
);
}
}
```

Notes

- Requires SMTP configuration in ZorroFix.ini
- SYNCHRONOUS - can delay (1-3 sec)
- Max 1 attachment per email
- Attachment: path relative to Zorro/Data
- Rate limit: ~10 emails/min (SMTP server)

SMTP Config (ZorroFix.ini)

```
ini
[SMTP]
Server = smtp.gmail.com:587
User = yourname@gmail.com
Password = yourapppassword
```

8.7.2. sound

Plays WAV audio files.

Syntax

```
void sound(string FileName);
```

Parameters

- **FileName** (string): Path file .wav

Return Value

Void

Example of Use

```
function run()
{
  // Sound alert on trade
if(NumOpenLong > NumOpenLong[1]) {
sound("Sounds/buy.wav");
}
else if(NumOpenShort > NumOpenShort[1]) {
sound("Sounds/sell.wav");
}

  // Error alert
if(ProfitClosed < -1000) {
sound("Sounds/alarm.wav");
}

  // Completion sound
if(is(EXITRUN)) {
sound("Sounds/complete.wav");
}
}
```

Notes

- WAV format only
- Does not block execution
- Silent if file not found
- Path relative to Zorro directory
- Max ~2 seconds typically

8.7.3. message

Show message box (dialog).

Syntax

```
int msg(string Text);
```

Parameters

- **Text** (string): Message to display

Return Value

int - 1 if OK, 0 if canceled

Example of Use

```
function run()
```

```
{
if(is(INITRUN)) {
    // Confirmation before live trade
if(is(LIVE)) {
if(!msg("Start LIVE trading?\nPress OK to continue")) {
quit("User cancelled");
}
}

    // Warning on config
if(StopLoss == 0) {
msg("WARNING: No stop loss defined!\n"
"This is dangerous in live mode.");
}
}

  // Critical alert
if(ProfitTotal < -5000) {
msg("CRITICAL: Loss exceeded $5000!\n"
      "Review strategy immediately.");
}
}
```

Notes

- ⚠ BLOCK execution until answered
- GUI mode only
- Not available in server/headless mode
- Useful for one-time confirmations/warnings

8.8. 📊 PLOTTING FUNCTIONS

Plotting functions enable advanced visualization of analysis, performance and patterns.

8.8.1. color

Defines color ranges for gradient plotting.

Syntax

void color(int Color1, int Color2, int Color3);

Parameters

- **Color1** , **Color2** , **Color3** (int): RGB colors per gradient

Example

```
// Gradient red->yellow->green
color(COLOR_RED, COLOR_YELLOW, COLOR_GREEN);

// Gradient plot
plot("Equity", Equity, LINE, COLOR_RANGE);
```

8.8.2. colorScale

Lightens/darkens color for variations.

Syntax

int colorScale(int Color, var Factor);

Parameters

- **Color** (int): RGB base color
- **Factor** (var): Multiplier (0.5=dark, 2.0=light)

Return

int - Color changed

Example

```
int baseColor = COLOR_BLUE;
int darkBlue = colorScale(baseColor, 0.5); // 50% darker
int lightBlue = colorScale(baseColor, 1.5); // 50% lighter
plotGraph("High", 0, darkBlue);
plotGraph("Low", 0, lightBlue);
```

8.8.3. plot

Plot time series on main chart.

Syntax

void plot(string Name, var Value, int Type, int Color);

Parameters

- **Name** (string): Series name (legend)
- **Value** (var): Value to plot
- **Type** (int): LINE, DOT, BAND, HISTOGRAM, etc.
- **Color** (int): RGB color

Example

```
function run()
{
  // Moving averages
```

```
plot("SMA20", SMA(series(priceClose()), 20), LINE, COLOR_BLUE);
plot("SMA50", SMA(series(priceClose()), 50), LINE, COLOR_RED);

  // Band
var upper = BBands(series(priceClose()), 20, 2);
var lower = BBands(series(priceClose()), 20, -2);
plot("BB_Upper", upper, BAND, COLOR_GRAY);
plot("BB_Lower", lower, BAND, COLOR_GRAY);

  // Histogram
var volume = marketVol(0);
plot("Volume", volume, HISTOGRAM, COLOR_CYAN);
}
```

Available Types

- LINE = Continuous line
- DOT = Points
- BAND = Band (filling)
- HISTOGRAM = Vertical bars
- MAIN = Overlays the price
- AXIS2 = Secondary Y-axis

8.8.4. plotBar

Colored bar plot on graph.

Syntax

void plotBar(string Name, int Num, var Value, int Type, int Color);

Parameters

- **Name** (string): Series name
- **Num** (int): Bar position (0=current bar, 1=previous, etc.)
- **Value** (var): Bar height
- **Type** , **Color** : Come plot()

Example

```
function run()
{
  // Performance-based color bars
var barReturn = (priceClose() - priceOpen()) / priceOpen();
int color = barReturn > 0 ? COLOR_GREEN : COLOR_RED;

plotBar("Returns", 0, barReturn * 100, HISTOGRAM, color);
```

```
   // Highlight specific bars
if(High(0) == HH(series(priceHigh()), 20)) {
plotBar("HighBars", 0, 1, DOT, COLOR_YELLOW);
   }
}
```

8.8.5. plotGraph

Plot symbol/marker on chart.

Syntax

void plotGraph(string Name, int Num, int Color);

Parameters

- **Name** (string): Symbol (arrow, cross, etc.)
- **Num** (int): Bar offset (0=current)
- **Color** (int): Symbol color

Example

```
function run()
{
   // Entry signals
if(crossOver(SMA1, SMA2)) {
plotGraph("Buy", 0, COLOR_GREEN);
}
else if(crossUnder(SMA1, SMA2)) {
plotGraph("Sell", 0, COLOR_RED);
}

   // Pivot points
if(peak(series(priceHigh()))) {
plotGraph("Peak", 0, COLOR_YELLOW);
   }
}
```

Default Symbols

- "Up" = Up arrow
- "Down" = Down arrow
- "Buy" = Buy Marker (green)
- "Sell" = Marker sell (red)
- "Cross" = Cross
- "Dot" = Point

- "Peak" = Peak Marker
- "Valley" = Valley Marker

8.8.6. plotChart

Update chart with new data.

Syntax

void plotChart(string Name, var x, var y, int Type, int Color);

Parameters

- **Name** (string): Series name
- **x** , **y** (var): Point coordinates
- **Type** , **Color** : Plot type and color

Example

```
function run()
{
  // Custom Scatter Plot
var rsi = RSI(series(priceClose()), 14);
var momentum = Mom(series(priceClose()), 10);

plotChart("RSI_Mom", rsi, momentum, DOT, COLOR_BLUE);

  // Equity curve custom
plotChart("PL", Bar, ProfitTotal, LINE, COLOR_GREEN);
}
```

8.8.7. plotData

Extracts plot data for export/analysis.

Syntax

string plotData(string Name, int Count);

Parameters

- **Name** (string): Series name to extract
- **Count** (int): Number of points (-1=all)

Return

string - CSV data format

Example

```
function run()
{
if(is(EXITRUN)) {
```

```
    // Export equity curve
string equityData = plotData("Equity", -1);
file_write("Export/equity.csv", equityData, 0);

    // Export indicator
string rsiData = plotData("RSI", 1000); // Last 1000
    file_write("Export/rsi.csv", rsiData, 0);
  }
}
```

8.8.8. plotBuyHold

Plot buy & hold strategy as benchmark.

Syntax

void plotBuyHold(int Mode);

Parameters

- **Mode** (int): 1=enable, 0=disable

Example

```
function run()
{
  if(is(INITRUN)) {
plotBuyHold(1); // Show benchmarks
  }

  // The strategy
  if(crossOver(SMA20, SMA50)) enterLong();
if(crossUnder(SMA20, SMA50)) exitLong();
}
```

8.8.9. plotContract

Plot payoff diagram for options.

Syntax

void plotContract();

Example

```
function run()
{
  // Short Strangle
```

```
contract(PUT, 4400, 30);
enterShort(1);

contract(CALL, 4600, 30);
enterShort(1);

  // Show payoff
plotContract();
}
```

8.8.10. plotCorrelogram

Plot autocorrelation series.

Syntax

```
void plotCorrelogram(string Name, vars Data, int Length);
```

Parameters

- **Name** (string): Graphic title
- **Data** (vars): Time series
- **Length** (int): Number of lags

Example

```
function run()
{
if(is(EXITRUN)) {
vars returns = series(ROC(series(priceClose()), 1));
      plotCorrelogram("Returns ACF", returns, 50);
}
}
```

8.8.11. plotDay / plotDayProfit

Intraday (hourly) performance profile.

Syntax

```
void plotDay(string Name, int Days);
void plotDayProfit();
```

Example

```
function run()
{
if(is(EXITRUN)) {
```

```
plotDay("Hourly Pattern", 60); // Last 60 days
    plotDayProfit(); // Profit by hour of day
}
}
```

8.8.12. plotHeatmap

Create heatmap matrix.

Syntax

```
void plotHeatmap(string Name, vars Data, int Width, int Height);
```

Example

```
function run()
{
if(is(EXITRUN)) {
    // Correlation matrix
vars corrMatrix = series(...); // Correlation matrix
plotHeatmap("Asset Correlations", corrMatrix, 10, 10);
}
}
```

8.8.13. plotHistogram

Distribution of values (histogram).

Syntax

```
void plotHistogram(string Name, vars Data, int Bins);
```

Example

```
function run()
{
if(is(EXITRUN)) {
vars returns = series(ROC(series(priceClose()), 1));
    plotHistogram("Returns Distribution", returns, 50);
}
}
```

8.8.14. plotMAEGraph / plotMAEPercentGraph

Maximum Adverse Excursion analysis.

Syntax

```
void plotMAEGraph(string Name);
```

```
void plotMAEPercentGraph(string Name);
```

Example

```
function run()
{
   if(is(EXITRUN)) {
plotMAEGraph("MAE Analysis");
      plotMAEPercentGraph("MAE %");
}
}
```

8.8.15. plotMFEGraph / plotMFEPercentGraph

Maximum Favorable Excursion analysis.

Syntax

```
void plotMFEGraph(string Name);
void plotMFEPercentGraph(string Name);
```

Example

```
function run()
{
   if(is(EXITRUN)) {
plotMFEGraph("MFE Analysis");
      plotMFEPercentGraph("MFE %");
}
}
```

8.8.16. plotMonth / plotMonthProfit

Monthly seasonality.

Syntax

```
void plotMonth(string Name, int Years);
void plotMonthProfit();
```

Example

```
function run()
{
if(is(EXITRUN)) {
plotMonth("Monthly Seasonality", 10);
      plotMonthProfit(); // Profit per month
}
```

```
}
```

8.8.17. plotPriceProfile

Price distribution (TPO/Volume Profile).

Syntax

```
void plotPriceProfile(string Name, int Bars);
```

Example

```
function run()
{
if(is(EXITRUN)) {
      plotPriceProfile("Price Distribution", 500);
   }
}
```

8.8.18. plotQuarterProfit

Profit per quarter.

Syntax

```
void plotQuarterProfit();
```

Example

```
function run()
{
if(is(EXITRUN)) {
plotQuarterProfit();
}
}
```

8.8.19. plotTradeProfile

Profit/loss trade distribution.

Syntax

```
void plotTradeProfile(string Name);
```

Example

```
function run()
{
   if(is(EXITRUN)) {
plotTradeProfile("Trade PL Distribution");
   }
```

```
}
```

8.8.20. plotWeek / plotWeekProfit

Weekly seasonality.

Syntax

```
void plotWeek(string Name, int Weeks);
void plotWeekProfit();
```

Example

```
function run()
{
if(is(EXITRUN)) {
plotWeek("Weekly Pattern", 52);
	plotWeekProfit();
}
}
```

8.8.21. plotWFOCycle / plotWFOProfit

Walk-Forward Optimization analysis.

Syntax

```
void plotWFOCycle();
void plotWFOProfit();
```

Example

```
function run()
{
  if(is(EXITRUN) && Optimize) {
plotWFOCycle(); // Cycle performance
	plotWFOProfit(); // Per-cycle profit
}
}
```

8.8.22. plotYear

Annual seasonality.

Syntax

```
void plotYear(string Name);
```

Example

```
function run()
```

```
{
if(is(EXITRUN)) {
plotYear("Yearly Performance");
}
}
```

8.9. PYTHON BRIDGE

8.9.1. pyStart

Initialize Python session.

Syntax

int pyStart(string PythonPath);

Parameters

- **PythonPath** (string): Path Python executable (NULL=auto)

Return

int - 1 success, 0 failure

Example

```
function run()
{
if(is(INITRUN)) {
if(!pyStart(NULL)) {
quit("Python not found!");
}

    // Import libraries
pyX("import numpy as np");
pyX("import pandas as pd");
    pyX("import sklearn");
}
}
```

8.9.2. pyX

Runs Python code.

Syntax

int pyX(string Code);

Parameters

- **Code** (string): Python code to execute

Return

int - 1 success, 0 failure

Example

```
function run()
{
  // Define Python function
pyX("def calculate_sharpe(returns):");
pyX(" return returns.mean() / returns.std()");

  // Setup data
pyX("prices = []");

  // Append data to each bar
pyX(strf("prices.append(%.4f)", priceClose()));

  // Calculate at end
if(is(EXITRUN)) {
pyX("sharpe = calculate_sharpe(pd.Series(prices).pct_change())");
var sharpe = pyVar("sharpe");
printf("\nPython Sharpe: %.2f", sharpe);
  }
}
```

8.9.3. pySet

Send variable/array from Zorro to Python.

Syntax

void pySet(string VarName, vars Data, int Length);

Parameters

- **VarName** (string): Python variable name
- **Data** (vars): Array Zorro
- **Length** (int): Number of elements

Example

```
function run()
{
if(is(EXITRUN)) {
    // Send prices to Python
vars prices = series(priceClose());
```

```
pySet("prices", prices, 252); // One year

    // Analyze with Python
pyX("import numpy as np");
pyX("returns = np.diff(prices) / prices[:-1]");
pyX("volatility = np.std(returns) * np.sqrt(252)");

var vol = pyVar("volatility");
printf("\nAnnual Volatility: %.2f%%", vol * 100);
  }
}
```

8.9.4. pyVar

Read float variable from Python.

Syntax

```
var pyVar(string VarName);
```

Example

```
var result = pyVar("sharpe_ratio");
var prediction = pyVar("model.prediction");
```

8.9.5. pyInt

Read integer variable from Python.

Syntax

```
int pyInt(string VarName);
```

Example

```
int bestPeriod = pyInt("optimal_period");
int clusterId = pyInt("cluster");
```

8.9.6. pyVec

Read arrays from Python.

Syntax

```
int pyVec(string VarName, vars Dest, int Length);
```

Parameters

- **VarName** (string): Python array name
- **Dest** (vars): Destination Zorro
- **Length** (int): Max elements

Return

int - Number of elements copied

Example

```
function run()
{
if(is(EXITRUN)) {
	// Python forecast
pyX("from statsmodels.tsa.arima.model import ARIMA");
pySet("prices", series(priceClose()), 252);
pyX("model = ARIMA(prices, order=(1,1,1))");
pyX("model_fit = model.fit()");
pyX("forecast = model_fit.forecast(steps=10)");

	// Read forecast
vars prediction = series(0);
int n = pyVec("forecast", prediction, 10);

printf("\nNext 10 days forecast:");
for(int i = 0; i < n; i++) {
printf("\nDay %d: %.2f", i+1, prediction[i]);
	}
}
}
```

8.10. 📈 R BRIDGE

8.10.1. Rstart

Initialize R session.

Syntax

int Rstart(string RPath, int Wait);

Parameters

- **RPath** (string): Path R executable
- **Wait** (int): Timeout milliseconds

Return

int - Session handle, 0 if error

Example

```
function run()
```

```
{
if(is(INITRUN)) {
if(!Rstart("C:/Program Files/R/R-4.1.0/bin/x64/R.exe", 5000)) {
quit("R not available");
}

    // Load libraries
Rx("library(quantmod)");
Rx("library(PerformanceAnalytics)");
  }
}
```

8.10.2. Rx

Runs R code.

Syntax

int Rx(string Code);

Example

```
function run()
{
  // Define R function
Rx("calculate_metrics <- function(returns) {");
Rx(" list(");
Rx(" sharpe = mean(returns) / sd(returns),");
  Rx(" sortino = SortinoRatio(returns)");
Rx(" )");
  Rx("}");

  // Use function
Rset("returns", returns_series, 252);
Rx("metrics <- calculate_metrics(returns)");

var sharpe = Rd("metrics$sharpe");
  var sortino = Rd("metrics$sortino");
}
```

8.10.3. Rset

Send array from Zorro to R.

Syntax

void Rset(string VarName, vars Data, int Length);

Example

```
vars prices = series(priceClose());
Rset("prices", prices, 252);
Rx("returns <- diff(prices) / prices[-length(prices)]");
```

8.10.4. Rd / Ri / Rv

Read variables from R.

Syntax

var Rd(string Expression); *// Double*

int Ri(string Expression); *// Integer*

int Rv(string VarName, vars Dest, int Length); *// Vector*

Example

```
// Scalars
var correlation = Rd("cor(x, y)");
int nRows = Ri("nrow(data)");

// Vectors
vars forecast = series(0);
int n = Rv("predictions", forecast, 30);
```

8.10.5. Rrun

Check R session status.

Syntax

int Rrun();

Return

int - 1 if R is active, 0 otherwise

Example

```
if(Rrun()) {
   // R available
Rx("analysis()");
}
else {
printf("\n!!! R session closed");
```

}

8.11. MISC UI FUNCTIONS

8.11.1. printf / print

Formatted output to console/log/file.

Syntax

void printf(string Format, ...);

void print(int Target, string Format, ...);

Parameters

- **Target** (int): TO_WINDOW, TO_FILE, TO_PRINTER, TO_CSVFILE
- **Format** (string): Format string (C printf style)
- **...** : Variable arguments

Example

```
function run()
{
  // Console
printf("\nBar %d: Price=%.2f PL=%.2f", Bar, priceClose(), ProfitTotal);

  // File
print(TO_FILE, "%s,%.2f,%.2f\n",
strdate("%Y-%m-%d", wdate()),
priceClose(),
Equity);

  // CSV export
print(TO_CSVFILE, "%.4f,%.4f,%.4f\n",
Open(0), High(0), Low(0));
}
```

Format Specifiers

%d, %i = int

%f = float (default 6 decimals)

%.2f = float with 2 decimal places

%s = string

%x = hexadecimal

%% = literal %

8.11.2. slider

Create UI sliders for user input.

Syntax

var slider(int Num, var Default, var Min, var Max, var Step, string Name);

Parameters

- **Num** (int): Slider ID (1-10)
- **Default** (var): Initial value
- **Min** , **Max** (var): Range
- **Step** (var): Increment
- **Name** (string): Label

Return

var - Current slider value

Example

```
function run()
{
  // 3 sliders for manual optimization
  var period = slider(1, 20, 10, 100, 5, "Period");
var stopLoss = slider(2, 20, 10, 50, 5, "Stop Loss");
var takeProfit = slider(3, 40, 20, 100, 10, "Take Profit");

  // Use slider values
Stop = stopLoss * PIP;
TakeProfit = takeProfit * PIP;
var sma = SMA(series(priceClose()), (int)period);
  if(priceClose() > sma) enterLong();
}
```

Notes

- Max 10 simultaneous sliders
- Values persist between runs
- Useful for interactive tuning
- Not available in TEST/TRADE mode

8.11.3. progress

Update progress bar.

Syntax

void progress(var Percent, string Text);

Parameters

- **Percent** (var): Progress 0-100
- **Text** (string): Optional message

Example

```
function run()
{
  // During optimization
if(Optimize) {
var percent = (Bar * 100.0) / NumBars;
progress(percent, strf("Testing... %.0f%%", percent));
}

  // During WFO
if(is(TRAINING)) {
var cycleProgress = (WFOCycle * 100.0) / NumWFOCycles;
progress(cycleProgress, strf("WFO Cycle %d/%d",
                        WFOCycle, NumWFOCycles));
}
}
```

8.11.4. report

Generate PDF performance reports.

Syntax

```
void report(string FileName);
```

Parameters

- **FileName** (string): Output file name (NULL=Report.pdf)

Example

```
function run()
{
if(is(EXITRUN)) {
    // Standard reports
report(NULL);

    // Custom Report
string timestamp = strdate("%Y%m%d_%H%M%S", wdate());
string fileName = strf("Reports/strategy_%s.pdf", timestamp);
report(fileName);
```

```
    // Email report
email("trader@example.com", "Daily Report",
        "Performance attached", "Report.pdf");
}
}
```

8.11.5. keys

Send keystrokes to window.

Syntax

void keys(int Handle, string Keys);

Parameters

- **Handle** (int): Window handle (from window())
- **Keys** (string): Keys to send

Example

```
function run()
{
  // External automation
int notepad = window("Notepad");
if(notepad) {
keys(notepad, "Hello from Zorro!{ENTER}");
}

  // Special keys
keys(hwnd, "{TAB}{TAB}{ENTER}"); // 2 TAB + ENTER
keys(hwnd, "^c"); // CTRL+C
keys(hwnd, "%{F4}"); // ALT+F4
}
```

Special Keys

{ENTER}, {TAB}, {SPACE}, {BACKSPACE}
{UP}, {DOWN}, {LEFT}, {RIGHT}
{F1}..{F12}
^ = CTRL
% = ALT
+ = SHIFT

8.11.6. mouse

Gets mouse position.

Syntax

int mouse(int* x, int* y, int* Button);

Parameters

- **x** , **y** (int*): Screen coordinates
- **Button** (int*): Button state (bitmask)

Return

int - 1 if mouse moved/clicked

Example

```
function run()
{
int x, y, button;

if(mouse(&x, &y, &button)) {
if(button & 1) {
printf("\nLeft click at [%d,%d]", x, y);
}

if(button & 2) {
printf("\nRight click at [%d,%d]", x, y);
      }
}
}
```

8.11.7. window

Find window by title.

Syntax

int window(string Title);

Parameters

- **Title** (string): Window title (partial OK)

Return

int - Window handle, 0 if not found

Example

```
function run()
{
  // Check if the app is open
   int bloomberg = window("Bloomberg Terminal");
```

```
if(!bloomberg) {
printf("\n!!! Bloomberg not running");
}

  // Send commands
int excel = window("Microsoft Excel");
if(excel) {
keys(excel, "^s"); // Save (CTRL+S)
  }
}
```

9. CHAPTER 9: USER-DEFINED CALLBACK FUNCTIONS

9.1. GENERAL INFORMATION

User Functions are optional features that the user can define to extend/customize Zorro's behavior. They are automatically called by the system at specific times.

9.1.1. CATEGORIES:

⚙ **Lifecycle (3):**

- main() - One-time global initialization
- run() - Main strategy loop
- cleanup() - Final cleaning

📊 **Optimization (3):**

- objective() - Custom fitness function
- parameters() - Optimization Setup
- evaluate() - Post-process results

✅ **Trading (4):**

- manage() - Trade micromanagement
- order() - Intercept orders
- tick() - Callback every tick
- tock() - Periodic callback

🔧 **Custom (5):**

- bar() - Custom bar construction
- click() - UI panel clicks
- callback() - Generic Events
- error() - Error handling
- neural() - ML custom

9.1.2. Best Practices

Lifecycle:

- ✅ Use main() for one-time global setup
- ✅ Put all logic in run()
- ✅ Always implement cleanup() for resources
- ✅ Check is(INITRUN) for initialization
- ✅ Check is(EXITRUN) for finalization

Optimization:

- ✅ Custom objective() for multi-criteria
- ✅ Combine metrics with weights
- ✅ Penalize unrealistic results
- ✅ Prefer robust vs max performance

Trading:

- ✅ Use manage() for dynamic stops
- ✅ Be VERY careful with order()
- ✅ Throttle tick() with Tock when possible
- ✅ Monitor performance in tock()

Error Handling:

- ✅ Always implement error() for production
- ✅ Log errors with context
- ✅ Send alerts on critical errors
- ✅ Backup state before quit

9.2. ⚙ LIFECYCLE FUNCTIONS

9.2.1. main

Entry point executed ONCE when Zorro starts (optional).

Syntax

```
function main()
{
// Global initialization
}
```

When She Gets Called

- AT THE START of Zorro, BEFORE any other function
- Only once per Zorro session
- It is not called for every run

Typical Usage

```
function main()
```

```
{
printf("\n====================================");
printf("\n TRADING SYSTEM v2.3");
printf("\n (c) 2025 Your Company");
printf("\n====================================");

// Check requirements
if(version() < 250) {
quit("Requires Zorro 2.50 or higher!");
  }

// Load global configuration
string config = file_content("Config/global.ini");
  if(config) {
// Parse global settings...
  }

// Initialize external connections
if(!pyStart(NULL)) {
    printf("\n!!! Python unavailable - ML disabled");
  }
}
```

Notes

- Optional - can be omitted
- Also run in -c (console only) mode
- It does not have access to trading variables (Asset, Bar, etc.)
- Useful for global one-time setup
- If present, run() is required

9.2.2. run

Main function performed at every bar/tick.

Syntax

```
function run()
{
// Trading logic here
}
```

When She Gets Called

- **Once per BAR** in TEST/TRAIN mode
- **Once per TICK** in LIVE mode (if Tock=0)
- **Periodically** in LIVE mode with Tock > 0

Typical Usage

```
function run()
{
// One-time initialization
   if(is(INITRUN)) {
StartDate = 20200101;
EndDate = 20241231;
BarPeriod = 60; // 1 hour bars

assetList("AssetsIB.csv");
plotBuyHold(1);
}

// Trading Logic
vars price = series(priceClose());
var sma20 = SMA(price, 20);
var sma50 = SMA(price, 50);

// Entry signals
if(crossOver(sma20, sma50)) {
enterLong();
}

if(crossUnder(sma20, sma50)) {
exitLong();
}

// Monitoring
if(frame() % 100 == 0) {
printf("\nBar %d: PL=%.2f, Equity=%.2f",
Bar, ProfitTotal, Equity);
}
```

```
// Final cleanup
if(is(EXITRUN)) {
report("Strategy_Report.pdf");
printf("\nFinal Sharpe: %.2f", Sharpe);
   }
}
```

Notes

- **REQUIRED** if there is no main()
- Full access to all trading variables
- Central logic of the strategy
- Called thousands/millions of times

Flag Municipalities

- if(is(INITRUN)) // First run
- if(is(EXITRUN)) // Last execution
- if(is(LOOKBACK)) // During lookback period
- if(is(LIVE)) // Live trading mode
- if(is(TRAINING)) // WFO training period

9.2.3. cleanup

Final cleanup performed ON EXIT (optional).

Syntax

```
function cleanup()
{
// Resource cleanup
}
```

When She Gets Called

- At the CLOSING of Zorro
- AFTER is(EXITRUN) in run()
- Also on quit() or errors

Typical Usage

```
function cleanup()
{
printf("\n\n====================================");
printf("\n STRATEGY SHUTDOWN");
printf("\n====================================");
```

```
// Close connections
   if(Rrun()) {
Rx("save.image('workspace.RData')");
}

// Backup results
string timestamp = strdate("%Y%m%d_%H%M%S", wdate());
string backup = strf("Backup/results_%s.txt", timestamp);
file_copy("Log/strategy.log", backup);

// Save state
saveStatus("last_state.dat");

// Notification
email("trader@example.com",
"Strategy Stopped",
strf("Final PL: %.2f", ProfitTotal),
        NULL);

printf("\nCleanup complete.");
}
```

Notes

- Optional but RECOMMENDED
- Guaranteed execution even in case of crash
- Does not have access to current bar data
- Useful for:
    - Save states
    - Close connections
    - Automatic backups
    - Final logging

9.3. 📊 OPTIMIZATION FUNCTIONS

9.3.1. objective

Defines objective function for optimization (optional).

Syntax

```
var objective()
{
```

```
// Calculate and return score
return score;
}
```

When She Gets Called

- At the END of each optimization run
- For EVERY parameter combination
- Only when Optimize = 1

Typical Usage

```
function objective()
{
// Custom fitness function
var score;

// Maximize Sharpe with drawdown penalty
score = Sharpe;

if(MaxDrawdown > 0.20) {
score -= 10.0; // Heavy penalty
}

// Penalty for few trades
   if(NumTrades < 100) {
score -= 5.0;
}

// Bonus for winning rate
if(WinTotal / NumWinTotal > LossTotal / NumLossTrades) {
score += 2.0;
}

// Multi-objective with weights
var robustness = (Sharpe + Sortino) / 2.0;
var efficiency = (ProfitTotal / abs(MaxDrawdown)) / 1000.0;
score = 0.6 * robustness + 0.4 * efficiency;
   return score;
}

function run()
```

```
{
// Optimize parameters
   var period = optimize(20, 10, 100, 10);
var stop = optimize(20, 10, 50, 5);

   // Strategy...
}
```

Notes

- If NOT defined, Zorro uses Sharpe Ratio
- HIGHER Score = better
- Can combine multiple metrics
- ALL performance variables accessible

Common Metrics

- Sharpe, Sortino // Risk-adjusted returns
- MaxDrawdown, AvgDrawdown // Risk metrics
- ProfitTotal, ProfitFactor// Profit metrics
- WinTotal, NumWinTotal // Win statistics
- UlcerIndex, CalmarRatio // Advanced metrics

9.3.2. parameters

Optimization parameters setup (optional - advanced).

Syntax

```
function parameters()
{
// Defines parameters to optimize
}
```

Typical Usage

```
function parameters()
{
// Defines range and step for optimize()
OptimalF = 1; // Kelly sizing

Verbose = 0; // Silent during opt

// Custom parameter names (for reports)
// ... advanced usage ...
}
```

Notes

- Rarely necessary
- For fine control on optimization
- Limited documentation - advanced use

9.3.3. evaluate

Post-process optimization results (optional).

Syntax

```
var evaluate(var* v, int n)
{
// v = array results optimization runs
// n = number of runs
// return = final score
}
```

Typical Usage

```
function evaluate(var* results, int numRuns)
{
// Calculate median instead of max
sortData(results, numRuns);
   int medianIdx = numRuns / 2;
var medianScore = results[medianIdx];
printf("\nMedian Score: %.2f", medianScore);
return medianScore;
}

// Or: Discard outliers
function evaluate(var* results, int numRuns)
{
var sum = 0;
int count = 0;

// Average excluding top/bottom 10%
int start = numRuns / 10;
int end = numRuns - start;
for(int i = start; i < end; i++) {
sum += results[i];
count++;
}
```

```
return sum / count;
}
```

Notes

- Very advanced - rarely used
- Default: max(results)
- Useful for statistical robustness

9.4. ✅ TRADING FUNCTIONS

9.4.1. manage

Trade micromanagement (optional).

Syntax

```
function manage()
{
// Edit open trades
}
```

When She Gets Called

- Every bar FOR EVERY trade open
- AFTER run() but BEFORE execution
- ThisTrade points to the current trade

Typical Usage

```
function manage()
{
// Trailing stop
   if(ThisTrade->fResult > 0) {
var newStop = priceClose() - 2 * ATR(14);

      if(newStop > ThisTrade->fStopPrice) {
ThisTrade->fStopPrice = newStop;
}
}

// Partial exit on profit target
if(ThisTrade->fResult > ThisTrade->fRisk * 2) {
ThisTrade->fAmount /= 2; // Exit 50%
}
```

```
// Time-based exit
var daysOpen = (wdate() - ThisTrade->tEntryDate);
if(daysOpen > 5) {
exitTrade(ThisTrade);
}

// Dynamic stop based on volatility
var currentATR = ATR(series(priceClose()), 14);
   ThisTrade->fStopPrice = priceClose() - 3 * currentATR;
}
```

Notes

- Call FOR EVERY trade
- Do not interfere with run() logic
- ThisTrade Global Access
- Useful for advanced trailing stops

Editable ThisTrade Fields

- ThisTrade->fStopPrice // Stop loss price
- ThisTrade->fTakeProfit // Take profit price
- ThisTrade->fAmount // Position size
- // Do NOT modify: fEntryPrice, tEntryDate

9.4.2. order

Custom order transmission (optional - advanced).

Syntax

```
int order(int OrderType)
{
// OrderType = operation opcode
// return = broker response
}
```

When She Gets Called

- For EVERY order sent to the broker
- Before the actual broadcast
- Allows interception/modification

Typical Usage

```
function order(int type)
{
// Log all orders
   if(type == ORD_ENTER) {
```

```
printf("\nEntering %s @ %.2f",
            ThisTrade->sSymbol,
ThisTrade->fEntryPrice);
}

// Modify order before sending
if(type == ORD_ENTER && is(LIVE)) {
// Round a tick size
ThisTrade->fEntryPrice = roundto(
          ThisTrade->fEntryPrice,
PIP
);
}

// Block orders under certain conditions
   if(hour() > 21 && is(LIVE)) {
printf("\n!!! Market closed - order blocked");
return 0; // Block
}

return 1; // Proceed normally
}
```

Order Types

- ORD_ENTER // Entry order
- ORD_EXIT // Exit order
- ORD_STOP // Stop loss triggered
- ORD_PROFIT // Take profit triggered

Notes

- VERY advanced - be careful!
- Return 0 = BLOCK order
- Return 1 = proceed normally
- Use with caution in LIVE

9.4.3. tick

Callback for EVERY tick market data (optional).

Syntax

```
function tick()
{
```

```
// Executed on every tick
}
```

When She Gets Called

- EVERY TICK in LIVE mode
- Not called in TEST (use run())
- Only with Tock = 0

Typical Usage

```
function tick()
{
// Scalping on tick data
static var lastBid = 0;
static var lastAsk = 0;

   var bid = priceBid();
var ask = priceAsk();

   // Detect bid/ask jump
   if(bid - lastBid > 5*PIP) {
printf("\n!!! Bid jump: %.5f -> %.5f", lastBid, bid);
// Possible immediate entry
}

// Monitor spread
var spread = ask - bid;
if(spread > 10*PIP) {
printf("\n!!! Wide spread: %.1f pips", spread/PIP);
exitLong(); // Exit for safety
}

lastBid = bid;
   lastAsk = ask;
}
```

Notes

- VERY FAST - optimizes code!
- Can call enterLong(), etc.
- Don't calculate heavy indicators
- Use Tock for throttling

9.4.4. tock

Periodic callback in LIVE (optional).

Syntax

```
function touch()
{
// Executed every Tock milliseconds
}
```

When She Gets Called

- Every Tock milliseconds in LIVE
- Alternative to tick() to avoid overloading
- Tock = 0 → use tick() instead

Typical Usage

```
function run()
{
if(is(INITRUN)) {
Tock = 1000; // Every 1 second
}
}

function touch()
{
// Update every second

// Position Monitor
if(NumOpenLong > 0) {
var currentPL = 0;
for(all_trades) {
currentPL += ThisTrade->fResult;
}

printf("\rPL: $%.2f", currentPL);
}

// Check margin
var marginUsed = brokerCommand(GET_MARGINUSED, 0);
var marginAvailable = brokerCommand(GET_MARGINAVAILABLE, 0);

if(marginUsed / (marginUsed + marginAvailable) > 0.80) {
```

```
printf("\n!!! Margin warning: 80%% used");
email("trader@example.com", "Margin Alert",
strf("Margin: %.0f%%", (marginUsed/(marginUsed+marginAvailable))*100),
        NULL);
}
}
```

Notes

- More efficient than tick()
- Tock=1000 → 1 call/sec instead of 100s/sec
- Use for monitoring, not quick entry
- Default Tock=0 (disabled)

9.5. CUSTOM FUNCTIONS

9.5.1. bar

Custom bar construction (optional - very advanced).

Syntax

```
BOOL bar(DATE* pTime, var* pHigh, var* pLow, var* pClose, var* pOpen, int Mode)
{
// Build custom bars
return TRUE/FALSE;
}
```

When She Gets Called

- For EVERY tick in bar formation
- Allows non-temporal bars (Renko, Range, etc.)

Typical Usage - Range Bars

```
static var rangeSize = 0;
static var barHigh, barLow, barOpen;
static BOOL barStarted = FALSE;

function bar(DATE* pTime, var* pH, var* pL, var* pC, var* pO, int mode)
{
if(!barStarted) {
// Start new bar
barOpen = barHigh = barLow = *pC;
barStarted = TRUE;
return FALSE; // Not yet complete
```

```
}

// Update high/low
barHigh = max(barHigh, *pH);
   barLow = min(barLow, *pL);

// Check if range reached
   if(barHigh - barLow >= rangeSize) {
// Full bar!
*pTime = wdate();
*pO = barOpen;
*pH = barHigh;
*pL = barLow;
*pC = *pC; // Current close

barStarted = FALSE;
return TRUE; // Bar ready
}

return FALSE; // Continue accumulation
}

function run()
{
if(is(INITRUN)) {
rangeSize = 10 * PIP; // 10 pip range bars
}

   // Range Bar Strategy...
}
```

Notes

- EXTREMELY ADVANCED
- Examples: Renko, Kagi, Point & Figure
- Requires deep understanding of bar mechanics

9.5.2. click

UI panel click callback (optional).

Syntax

```
void click(int ID, int Row, int Col)
{
// Manage clicks on panel cell
}
```

When She Gets Called

- When user clicks cell panel
- ID = panel ID, Row/Col = coordinates

Typical Usage

```
function run()
{
if(is(INITRUN)) {
// Create control panel
panel(1, 5, 2, 150, 30, 0);

panelSet(1, 0, 0, "START", 0, COLOR_GREEN);
panelSet(1, 1, 0, "STOP", 0, COLOR_RED);
panelSet(1, 2, 0, "RESET", 0, COLOR_YELLOW);
}
}

function click(int panelID, int row, int col)
{
if(panelID == 1) {
if(row == 0) {
// START clicked
printf("\nStarting strategy...");
Live = 1;
}
else if(row == 1) {
// STOP clicked
printf("\nStopping...");
Live = 0;
exitLong();
exitShort();
}
else if(row == 2) {
```

```
// RESET clicked
printf("\nResetting...");
saveStatus(NULL); // Clear
      }
}
}
```

9.5.3. callback

Generic callback for external events (optional).

Syntax

```
int callback(int ID, var Value1, var Value2)
{
// Handle generic callback
return status;
}
```

When She Gets Called

- From API broker events
- From external timers
- Custom triggers

Typical Usage

```
function callback(int eventID, var param1, var param2)
{
switch(eventID) {
case EVENT_TICK:
// New tick arrived
break;

case EVENT_FILL:
printf("\nOrder filled!");
break;

case EVENT_REJECT:
printf("\n!!! Order rejected: %.0f", param1);
break;

case EVENT_DISCONNECT:
```

```
printf("\n!!! Broker disconnected");
email("trader@example.com",
"Broker Disconnect",
"Connection lost!",
NULL);
        break;
}

return 1;
}
```

9.5.4. error

Custom error handler (optional).

Syntax

```
void error(string Message)
{
// Handle error
}
```

When She Gets Called

- On any runtime error
- Before termination (if fatal)

Typical Usage

```
function error(string msg)
{
printf("\n");
printf("\n=======================================");
printf("\n!!! ERROR OCCURRED !!!");
printf("\n=======================================");
printf("\n%s", msg);
printf("\n=======================================");

   // Detailed log
string logEntry = strf(
      "\n[%s] ERROR: %s\n"
"Bar: %d, Asset: %s, Time: %s\n"
"PL: %.2f, Equity: %.2f\n",
strdate("%Y-%m-%d %H:%M:%S", wdate()),
```

```
msg,
Bar, Asset, strdate("%Y-%m-%d", wdate()),
Profit Total, Equity
);

file_append("Errors/error_log.txt", logEntry);

   // Instant notification
email("trader@example.com",
        "Strategy Error!",
msg,
NULL);

// Backup state
saveStatus("emergency_backup.dat");

// Screenshot (if possible)
   // ...
}
```

9.5.5. neural

Custom ML/Neural Network (optional - advanced).

Syntax

```
var neural(int Mode, int Model, int NumSignals, var* Signals)
{
// Custom ML prediction
   return prediction;
}
```

When She Gets Called

- From adviseLong()/adviseShort()
- Mode: NEURAL_TRAIN or NEURAL_PREDICT

Typical Usage with Python

```
function neural(int mode, int model, int numInputs, var* inputs)
{
if(mode == NEURAL_TRAIN) {
// Training
pySet("features", inputs, numInputs);
```

```
pySet("target", series(priceClose()), 1);
pyX("from sklearn.ensemble import RandomForestRegressor");
	pyX("model = RandomForestRegressor(n_estimators=100)");
	pyX("model.fit(features.reshape(-1,1), target)");
return 1; // Training ok
}
else if(mode == NEURAL_PREDICT) {
// Prediction
pySet("input_data", inputs, numInputs);
pyX("prediction = model.predict(input_data.reshape(1,-1))");
var prediction = pyVar("prediction");
return forecast;
}
return 0;
}

function run()
{
// Use ML for entry
vars features = series(SMA(Close, 20));
var signal = adviseLong(NEURAL_TRAIN, 0,
	features, 1
);
if(signal > 0.6) {
enterLong();
}
}
```

Notes

- Requires deep ML knowledge
- Very useful Python/R bridge
- Limited documentation