



WealthScript Function Reference

Wealth-Lab Developer 4.0

© 2003-2006 WL Systems, Inc.

Wealth-Lab Developer 4.0 WealthScript Function Reference

by WL Systems, Inc.

Revised: Monday, December 11, 2006

Wealth-Lab Developer 4.0 WealthScript Function Reference

© 2003-2006 WL Systems, Inc.

No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Third party trademarks and service marks are the property of their respective owners.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use or misuse of information contained in this document or from the use or misuse of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: Monday, December 11, 2006

Special thanks to:

Wealth-Lab's great on-line community whose comments have helped make this manual more useful for veteran and new users alike.

EC Software, whose product HELP & MANUAL printed this document.

Table of Contents

Part I Introduction	13
Part II Alert Functions	15
1 Overview	15
2 AlertCount	16
3 AlertOrderType	16
4 AlertPositionType	17
5 AlertPrice	17
6 AlertShares	18
7 AlertSymbol	18
Part III Cosmetic Chart Functions	19
1 Overview	19
2 AnnotateBar	19
3 AnnotateChart	20
4 CreatePane	20
5 DrawCircle	20
6 DrawCircle2	21
7 DrawDiamond	21
8 DrawEllipse	22
9 DrawHorzLine	22
10 DrawImage	23
11 DrawLabel	23
12 DrawLine	24
13 DrawRectangle	24
14 DrawRoundRect	25
15 DrawText	25
16 DrawTriangle	26
17 EnableNotes	26
18 EnableTradeNotes	27
19 HidePaneLines	27
20 HideVolume	27
21 PlotSeries	28
22 PlotSeriesLabel	28
23 PlotStops	29
24 PlotSymbol	30
25 PlotSyntheticSymbol	30
26 SetBackgroundColor	31
27 SetBarColor	32
28 SetBarColors	32

29	SetColorScheme	33
30	SetLogScale	33
31	SetPaneBackgroundColor	33
32	SetPaneMinMax	34
33	SetSeriesBarColor	34
Part IV CommissionScript Functions		36
1	Overview	36
2	CMShares	36
3	CMPrice	36
4	CMAEntry	37
5	CMOrderType	37
6	CMSymbol	38
7	CMDataSource	38
8	CMResult	38
Part V Data Access Functions		39
1	Overview	39
2	BarCount	39
3	GetDate	39
4	GetMargin	40
5	GetPointValue	40
6	GetSecurityName	40
7	GetSymbol	41
8	GetTick	41
9	GetTime	42
10	OpenInterest	42
11	PriceAverage	43
12	PriceAverageC	43
13	PriceClose	44
14	PriceHigh	44
15	PriceLow	44
16	PriceOpen	45
17	Volume	45
Part VI Date/Time Functions		46
1	Overview	46
2	BarInterval	46
3	BarNum	46
4	CurrentDate	47
5	CurrentTime	47
6	DateTimeToBar	47
7	DateToBar	48

8	DateToStr	48
9	DayOfWeek	49
10	DaysBetween	49
11	DaysBetweenDates	50
12	GetDay	50
13	GetHour	51
14	GetMinute	51
15	GetMonth	52
16	GetYear	52
17	IsLeapYear	52
18	LastBar	53
19	OptionExpiryDate	54
20	StrToDate	54
21	StrToTime	55
22	TimeToStr	55

Part VII File Access Functions 56

1	Overview	56
2	FileClear	56
3	FileClose	56
4	FileCreate	56
5	FileEOF	57
6	FileFlush	57
7	FileOpen	57
8	FileRead	58
9	FileWrite	58

Part VIII Fundamental Data Access Functions 59

1	FundamentalPriceSeriesAverage	59
2	GetFundamentalDetail	59

Part IX Math Functions 61

1	Overview	61
2	Abs	61
3	ArcCos	61
4	ArcSin	62
5	ArcSinh	62
6	ArcTan	62
7	ArcTanh	62
8	Correlation	62
9	Cos	63
10	Cosh	63
11	Cotan	63

12	Dec	63
13	DegToRad	64
14	Exp	64
15	Frac	64
16	Hypot	64
17	Inc	64
18	Int	65
19	LinearRegLine	65
20	LineExtendX	66
21	LineExtendY	66
22	LN	67
23	Log10	67
24	Log2	67
25	Max	68
26	Min	68
27	Pi	69
28	Power	69
29	RadToDeg	70
30	RandG	70
31	Random	71
32	RandomInt	71
33	Randomize	71
34	RandSeed	72
35	Round	72
36	SetRandSeed	72
37	Sin	72
38	Sinh	73
39	Sqr	73
40	Sqrt	73
41	Tan	73
42	Tanh	73
43	TrendLineValue	73
44	Trunc	74
Part X PerfScript Functions		75
1	Overview	75
2	AccountExposure	75
3	CashInterest	76
4	DividendsPaid	76
5	MarginLoan	76
6	PerfAddCurrency	77
7	PerfAddNumber	77
8	PerfAddPct	78

9	PerfAddString	78
10	PerfAddBreak	79
11	StartingCapital	79
12	TotalCommission	79
Part XI Position Management Functions		80
1	Overview	80
2	ActivePositionCount	80
3	ClearPositions	81
4	GetPositionData	82
5	GetPositionPriority	83
6	GetPositionRiskStop	83
7	LastActivePosition	84
8	LastLongPositionActive	85
9	LastPosition	85
10	LastPositionActive	86
11	LastShortPositionActive	87
12	MarketPosition	87
13	PositionActive	88
14	PositionBasisPrice	89
15	PositionBarsHeld	90
16	PositionCount	90
17	PositionEntryBar	91
18	PositionEntryPrice	92
19	PositionExitBar	93
20	PositionExitPrice	94
21	PositionExitSignalName	94
22	PositionLong	95
23	PositionMAE	96
24	PositionMAEPct	96
25	PositionMFE	97
26	PositionMFEPct	98
27	PositionOpenMAE	99
28	PositionOpenMAEPct	99
29	PositionOpenMFE	100
30	PositionOpenMFEPct	100
31	PositionOpenProfit	101
32	PositionOpenProfitPct	102
33	PositionOrderType	102
34	PositionProfit	103
35	PositionProfitPct	104
36	PositionShares	105
37	PositionShort	105

38	PositionSignalName	106
39	PositionSymbol	107
40	SetPositionData	107
41	SetPositionPriority	108
42	SetPositionRiskStop	110
43	SetRiskStopLevel	110
Part XII Price Series Functions		112
1	Overview	112
2	AbsSeries	112
3	AddCalendarDays	112
4	AddFutureBars	113
5	AddSeries	114
6	AddSeriesValue	114
7	AnalyzeSeries	114
8	ChangeBar	115
9	ClearExternalSeries	116
10	ClearIndicators	116
11	CreateNamedSeries	117
12	CreateSeries	117
13	CreateSeriesLength	118
14	CrossOver	118
15	CrossOverValue	119
16	CrossUnder	119
17	CrossUnderValue	120
18	DivideSeries	120
19	DivideSeriesValue	120
20	DivideValueSeries	121
21	EnableSynch	121
22	FindNamedSeries	122
23	FirstActualBar	122
24	GetDescription	123
25	GetExternalSeries	123
26	GetSeriesValue	124
27	MultiplySeries	125
28	MultiplySeriesValue	125
29	OffsetSeries	125
30	RestorePrimarySeries	126
31	SetDescription	126
32	SetPrimarySeries	127
33	SetSeriesValue	128
34	SingleCalcMode	128
35	SubtractSeries	129

36	SubtractSeriesValue	130
37	SubtractValueSeries	130
38	SynchAll	130
39	SynchSeries	131
40	SyntheticBar	132
41	TurnDown	132
42	TurnUp	133
Part XIII SimuScript Functions		134
1	Overview	134
2	BarCount	134
3	BuyAndHold	134
4	CandidateCount	135
5	Cash	136
6	DrawDown	136
7	DrawDownPct	137
8	Equity	137
9	SetPositionSizeFixed	138
10	SetPositionSizePct	138
11	SetPositionSizeShares	138
12	SortByEntryDate	139
13	SortByExitDate	140
Part XIV String Functions		141
1	Overview	141
2	CharAt	141
3	Chr	142
4	CompareStr	142
5	CompareText	142
6	Copy	143
7	Delete	143
8	FloatToStr	143
9	FormatFloat	144
10	GetToken	144
11	Insert	145
12	IntToStr	145
13	Length	145
14	LowerCase	146
15	Ord	146
16	Pos	146
17	StrToFloat	147
18	StrToFloatDef	147
19	StrToInt	147

20	StrToIntDef	148
21	Trim	148
22	TrimLeft	148
23	TrimRight	149
24	UpperCase	149
Part XV System Functions		150
1	Overview	150
2	Abort	150
3	AddCommentary	150
4	AddScanColumn	151
5	AddScanColumnStr	151
6	AllowSymbolSearch	152
7	CreateOleObject	152
8	GetGlobal	153
9	GetScriptName	153
10	GetTickCount	154
11	Input	154
12	IWealthLabAddOn3	155
13	IWealthLabAuto	155
14	IsRealTime	156
15	Null	156
16	PlaySound	156
17	Print	157
18	PrintFlush	157
19	PrintStatus	158
20	RunProgram	158
21	SaveChartImage	159
22	SetGlobal	159
23	SetOptimizeValue	160
24	SetPeakTroughMode	161
25	ShowMessage	162
26	Sleep	162
27	UseUpdatedEMA	163
28	WatchListAddSymbol	163
29	WatchListClear	164
30	WatchListCount	165
31	WatchListDelete	165
32	WatchListName	166
33	WatchListRemoveSymbol	166
34	WatchListSelect	166
35	WatchListSymbol	167

Part XVI	Technical Indicator Functions	168
1	Overview	168
2	AccumDist	168
3	ADX	169
4	ADXR	170
5	AroonDown	171
6	AroonUp	172
7	ATR	173
8	ATRP	174
9	BBandLower	174
10	BBandUpper	175
11	BOP	176
12	CADO	176
13	CCI	178
14	CMF	179
15	CMO	180
16	CumDown	181
17	CumUp	182
18	DIMinus	183
19	DIPlus	184
20	DSS	185
21	DX	186
22	EMA	187
23	EMMinus	189
24	EMPlus	190
25	FAMA	190
26	FIR	191
27	Highest	192
28	HighestBar	193
29	HTDCPhase	194
30	HTInPhase	195
31	HTLeadSin	195
32	HTPeriod	197
33	HTQuadrature	198
34	HTSin	199
35	HTTrendLine	200
36	HV	202
37	Kalman	203
38	KAMA	204
39	KeltnerLower	205
40	KeltnerUpper	206
41	LinearReg	206

42	LinearRegPredict	207
43	LinearRegSlope	207
44	Lowest	208
45	LowestBar	209
46	MACD	209
47	MAMA	211
48	Median	212
49	MFI	212
50	Momentum	214
51	MomentumPct	215
52	MoneyFlow	215
53	NVI	216
54	OBV	217
55	Parabolic	218
56	Peak	220
57	PeakBar	221
58	PeakNum	222
59	PVI	223
60	QStick	224
61	RelSlope	225
62	ROC	225
63	RSI	227
64	RSquared	228
65	RVI	229
66	SMA	230
67	StdDev	231
68	StdError	232
69	StochD	233
70	StochK	234
71	StochRSI	235
72	Sum	236
73	TII	237
74	TRIX	238
75	Trough	239
76	TroughBar	240
77	TroughNum	240
78	TrueRange	242
79	UltimateOsc	243
80	VHF	243
81	Vidya	244
82	VMA	245
83	Volatility	246

84	WilderMA	247
85	WilliamsR	248
86	WMA	249

Part XVII Time Frame Functions 251


1	Overview	251
2	ChangeScale	251
3	DailyFromMonthly	252
4	DailyFromWeekly	252
5	GetDailyBar	253
6	GetIntraDayBar	253
7	GetMonthlyBar	254
8	GetWeeklyBar	254
9	IntraDayFromCompressed	255
10	IntraDayFromDaily	256
11	IsDaily	256
12	IsIntraday	256
13	IsMonthly	257
14	IsWeekly	257
15	SetScaleCompressed	257
16	SetScaleDaily	258
17	SetScaleMonthly	259
18	SetScaleWeekly	259

Part XVIII Trading System Control Functions 261

1	Overview	261
2	ApplyAutoStops	261
3	BuyAtClose	263
4	BuyAtLimit	263
5	BuyAtMarket	264
6	BuyAtStop	265
7	CoverAtClose	265
8	CoverAtLimit	266
9	CoverAtMarket	267
10	CoverAtStop	267
11	CoverAtTrailingStop	268
12	InstallBreakEvenStop	269
13	InstallProfitTarget	269
14	InstallReverseBreakEvenStop	270
15	InstallStopLoss	270
16	InstallTimeBasedExit	271
17	InstallTrailingStop	272
18	PortfolioSynch	273

19	SellAtClose	274
20	SellAtLimit	275
21	SellAtMarket	275
22	SellAtStop	276
23	SellAtTrailingStop	277
24	SetAutoStopMode	278
25	SetCommission	278
26	SetPositionSize	279
27	SetShareCap	279
28	SetShareFloor	280
29	SetShareSize	280
30	SetSlippage	281
31	ShortAtClose	281
32	ShortAtLimit	282
33	ShortAtMarket	283
34	ShortAtStop	283
35	SplitPosition	284
	Index	0

1 Introduction

The Function Reference defines, describes, and demonstrates the WealthScript functions by example. However, if you need more examples of a particular function, you can use the function search feature  of the ChartScript Explorer to find scripts that contain a specific WealthScript function.

Each function contains a header in its description that indicates if its use is valid for a particular type of script. For example, the following header indicates that the function is valid for use in ChartScripts and SimuScripts, but not in PerfScripts or CMScripts.

ChartScripts SimuScripts PerfScripts CMScripts

The legend below provides definitions for additional indicative symbology:

<input checked="" type="checkbox"/>	Valid for use
<input type="checkbox"/>	Invalid usage
<input checked="" type="checkbox"/>	Usage difference between ChartScripts and SimuScripts
<input checked="" type="checkbox"/>	Valid in specific cases

Useful tips:

1. Use the **QuickRef**, which is found in the main icon bar on the left. Place the cursor on a WealthScript function, which is syntax highlighted in blue by default, in the Editor view and press **F1** to call up the QuickRef for the function.
2. When coding manually, use the Smart Code Editor features. Before typing a WealthScript function name, strike **Ctrl+Space bar** to bring up a list of WealthScript functions. As you continue to type characters, you can filter the list to quickly locate the function you're looking for. Also, for functions with parameter lists, after typing the opening parenthesis "(" a list of parameters will be displayed with a cue for the current parameter in bold type.

WealthScript functions are found in at least one of 14 categories. Click below to be taken to an overview.

[Alert Functions](#) ^[15]

[Cosmetic Chart Functions](#) ^[19]

[CommissionScript Functions](#) ^[36]

[Data Access Functions](#) ^[39]

[Date/Time Functions](#) ^[46]

[File Access Functions](#) ^[56]

[Math Functions](#) ^[67]

[PerfScript Functions](#) ^[73]

[Position Management Functions](#) ^[80]

[Price Series Functions](#) ^[112]

[SimuScript Functions](#) ^[134]

[String Functions](#) ^[147]

[System Functions](#) ^[150]

[Technical Indicator Functions](#) ^[168]

[Time Frame Functions](#) ^[257]

[Trading System Control Functions](#) ²⁶⁷

2 Alert Functions

2.1 Overview

An **Alert** is an order that needs to be placed for the next bar. Using the Alert Functions, you can access the number of alerts that a script has generated, as well as the symbol, order type, position type, price, and number of shares (contracts) of a specific Alert.

Note: The Alert category of WealthScript functions are not available for SimuScripts.

The following example shows how you can create a text file of Alert information automatically from within any script.

Example

```
{ These declarations may appear at the beginning of the script }
const delim = '|';
const file = 'C:\Alerts.txt';
var MyAlert: string;
var a, FleHdl: integer;

{ A function to round Price to precisely 2 digits after the decimal }
function StockFix( Price: float ): float;
begin
  const factor = 100; // 1000 for 3 digits, etc.
  Result := Round( Price * factor ) / factor
end;

{ (* Your script's main body goes here *) }

if AlertCount > 0 then
begin
  FleHdl := FileOpen( file );
  for a := 0 to AlertCount - 1 do
  begin
    MyAlert := GetSymbol + delim
              + IntToStr( AlertPositionType( a ) ) + delim
              + IntToStr( AlertShares( a ) ) + delim
              + IntToStr( AlertOrderType( a ) ) + delim
              + FloatToStr( StockFix( AlertPrice( a ) ) );

    FileWrite( FleHdl, MyAlert );
  end;
end;
```

2.2 AlertCount

AlertCount: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the number of Alerts that have been generated. An Alert is an order that needs to be placed for the next bar. Use **AlertShares**, **AlertPositionType**, **AlertOrderType**, **AlertSymbol** and **AlertPrice** to gain more information on a specific Alert.

Example

```
{ Place at the end of your script }
if AlertCount > 0 then
  ShowMessage( IntToStr( AlertCount ) + ' Alert(s) for the next Bar!'
);
```

2.3 AlertOrderType

AlertOrderType(Alert: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the order type of the specified Alert. The *Alert* parameter is an integer value that contain an Alert Index number. The index number should be between zero (first Alert) and **AlertCount** - 1.

The return value will be one of the following:

- 0 = Market Order
- 1 = Stop Order
- 2 = Limit Order
- 3 = AtClose Order

Example

```
{ Place at the end of your script }
var a: integer;
var s: string;
for a := 0 to AlertCount - 1 do
begin
  s := 'Alert ' + IntToStr( a + 1 ) + ' is a';
  case AlertOrderType( a ) of
    0:
      s := s + ' Market';
    1:
      s := s + ' Stop';
    2:
      s := s + ' Limit';
    3:
      s := s + 'n At Close';
  end;
  s := s + ' Order';
  DrawLabel( s, 0 );
end;
```

2.4 AlertPositionType

AlertPositionType(Alert: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Position type of the specified Alert. The *Alert* parameter is an integer value that contain an Alert Index number. The index number should be between zero (first Alert) and **AlertCount** - 1.

The return value will be one of the following:

- 0 = Buy
- 1 = Sell
- 2 = Sell Short
- 3 = Cover Short

Example

```
{ Place at the end of your script }
var a: integer;
var s: string;
for a := 0 to AlertCount - 1 do
begin
  s := 'Alert ' + IntToStr( a + 1 ) + ' is a ';
  case AlertPositionType( a ) of
    0:
      s := s + 'Buy';
    1:
      s := s + 'Sell';
    2:
      s := s + 'Short';
    3:
      s := s + 'Cover';
  end;
  s := s + ' Order';
  DrawLabel( s, 0 );
end;
```

2.5 AlertPrice

AlertPrice(Alert: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the price of the specified *Alert*. Limit and Stop Order Alerts will have a Price only. The *Alert* parameter is an integer value that contain an Alert Index number. The index number should be between zero (first Alert) and **AlertCount** - 1.

Example

```
{ Place at the end of your script }
var a: integer;
for a := 0 to AlertCount - 1 do
  if ( AlertOrderType( a ) = 1 ) or ( AlertOrderType( a ) = 2 ) then
    DrawLabel( 'Alert ' + IntToStr( a + 1 ) + ' has a price of '
      + FormatFloat( '$#,##0.00', AlertPrice( a ) ), 0 );
```

2.6 AlertShares

AlertShares(Alert: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the number of shares, or contracts, in the specified *Alert*. The *Alert* parameter is an integer value that contain an Alert Index number. The index number should be between zero (first Alert) and **AlertCount** - 1.

Example

```
{ Place at the end of your script }
var a: integer;
for a := 0 to AlertCount - 1 do
  DrawLabel( 'Alert ' + IntToStr( a + 1 ) + ' is for '
            + IntToStr( AlertShares( a ) ) + ' shares', 0 );
```

2.7 AlertSymbol

AlertSymbol(Alert: integer): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the symbol of the specified Alert. The *Alert* parameter is an integer value that contain an Alert Index number. The index number should be between zero (first Alert) and **AlertCount** - 1.

Remarks

- Since you can make trades on symbols other than the 'clicked' symbol through the use of the **SetPrimarySeries** function, the **AlertSymbol** may not be the same as the 'clicked' symbol.

Example

```
{ Place at the end of your script }
var a: integer;
for a := 0 to AlertCount - 1 do
  DrawLabel( 'Alert ' + IntToStr( a + 1 ) + ' is for symbol: '
            + AlertSymbol( a ), 0 );
```

3 Cosmetic Chart Functions

3.1 Overview

In Wealth-Lab Developer 4.0, you have control over almost everything that is displayed on the chart. Whether it be a text annotation, a graphics object, or even a bitmap image, look towards the Cosmetic Chart Functions to do the job. Many of the functions use the color and style constants found here.

Note: The Cosmetic Chart category of WealthScript functions are not available for SimuScripts.

Color value constants (Color parameter)

#Black, #Maroon, #Green, #Olive, #Navy, #Purple, #Teal, #Gray, #Silver, #Red, #Lime, #Yellow, #Blue, #Fuchsia, #Aqua, #White, and finally #WinLoss, which is used primarily for [PerfScripts](#)^[75].

Light colors, normally used for shading the chart background:

#RedBkg, #BlueBkg, #GreenBkg

Additionally, colors can be specified as 3-digit integers representing a **RGB** color, where the first digit is the red color contribution, the second digit from green, and the third digit from blue. For example, **900** would be red only, whereas **009** is blue only.

Plot formatting (line Style parameter) constants

#Thin, #Dotted, #Thick, #Histogram, #ThickHist, #Dots

Finally, note that many default Chart settings are found in the Options dialog, **Tools|Options (F12)|Colors/Style** tab.

3.2 AnnotateBar

AnnotateBar(Text: string; Bar: integer; AbovePrices: boolean; Color: integer; FontSize: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Annotates the specified *Bar* with the text provided in the *Text* parameter. If *AbovePrices* is true, **AnnotateBar** draws the text above the bar, otherwise below it. Call **AnnotateBar** multiple times for the same *Bar* to stack *Text* strings above or below the bar.

Example

```
{ Annotate a bar if it's a 200 day closing low }
var BAR: integer;
for Bar := 20 to BarCount - 1 do
begin
  if PriceClose( Bar ) = Lowest( Bar, #Close, 200 ) then
    AnnotateBar( 'Low', Bar, false, #Black, 7 );
end;
```

3.3 AnnotateChart

AnnotateChart(Text: string; Pane, Bar: integer; Price: float; Color: integer; FontSize: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Annotates the chart with the specified *Text* at the position determined by the *Bar* and *Price* parameters. Use this function to draw ad-hoc annotations anywhere on the chart.

Example

```
{ Annotate the last bar if we have a support level below it }
var LP, P: float;
lp := Peak( BarCount - 1, #High, 6 );
p := PriceClose( BarCount - 1 );
if p > lp then
  AnnotateChart( 'Support', 0, PeakBar( BarCount - 1, #High, 6 ), lp,
    #Green, 8 );
```

3.4 CreatePane

CreatePane(Height: integer; AbovePrices: boolean; ShowGrid: boolean): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Creates a new blank chart pane. You can plot indicators in the pane using the **PlotSeries** function. The *Height* parameter specifies the height of the new pane in pixels. An average height of 75 pixels is common for new panes. The *AbovePrices* parameter specifies whether to draw the new pane above or below the main price/volume panes. The *Grid* parameter controls whether default horizontal grid lines are drawn on the pane.

Example

```
{ Create a new Pane and plot an RSI in it }
var MyPane: integer;
MyPane := CreatePane( 100, true, true );
PlotSeries( RSISeries( #Close, 30 ), MyPane, #Navy, #Thin );
```

3.5 DrawCircle

DrawCircle(Radius: integer; Pane: integer; Bar: integer; Price: float; Color: integer; Style: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Draws a circle with the specified *Radius* at a location determined by the values of the *Bar* and *Price* parameters.

Example

```
{ Circle any 200 day High }
var BAR: integer;
for Bar := 200 to BarCount - 1 do
begin
  if PriceHigh( Bar ) = Highest( Bar, #High, 200 ) then
```

```

    DrawCircle( 4, 0, Bar, PriceHigh( Bar ), #Red, #Thick );
end;

```

3.6 DrawCircle2

DrawCircle2(BarCenter: integer; PriceCenter: float; BarRadius: integer; PriceRadius: float; Pane: integer; Color:integer; Style: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Draws a circle centered on *BarCenter/PriceCenter*, and intersecting at point *BarRadius/PriceRadius*. The radius of the circle becomes the distance between these two points.

Example

```

var Bar1, Bar2, Radius: integer;
var x1, x2, y1, y2: float;

Bar1 := BarCount - 150;
Bar2 := BarCount - 100;
SetBarColor( Bar1, #Blue );
SetBarColor( Bar2, #Blue );

y1 := PriceClose( Bar1 );
y2 := PriceClose( Bar2 );

DrawCircle2( Bar1, y1, Bar2, y2, 0, #Red, #Thin );

```

3.7 DrawDiamond

DrawDiamond(Bar1: integer; Price1: float; Bar2: integer; Price2: float; Bar3: integer; Price3: float; Bar4: integer; Price4: float; Pane: integer; Color: integer; Style: integer; FillColor: integer; BehindPrices: boolean);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Draws a diamond (or any 4 sided polygon), the four corners of which are specified by the parameters *Bar1/Price1*, *Bar2/Price2*, *Bar3/Price3* and *Bar4/Price4*.

- | | |
|---------------------|---|
| <i>Pane</i> | Specifies which on which pane to draw the diamond. Pass 0 for the Price Pane, 1 for the Volume Pane, or use the return value of a CreatePane call for a custom pane. |
| <i>Color</i> | Controls the color used to draw the diamond. |
| <i>Style</i> | Controls the type of line used. You can use the constants #Thin , #Thick or #Dotted for <i>Style</i> . |
| <i>FillColor</i> | Specifies the color with which to fill the diamond. Pass -1 to draw an unfilled diamond. |
| <i>BehindPrices</i> | Controls whether the diamond is drawn behind or in front of the price bars. |

Example

```

var P1, P2, P3, P4: float;
var B, B3, B2, B1, B4: integer;
b := BarCount - 1;

```



```

b3 := TroughBar( b, #Close, 13 );
b2 := PeakBar( b3, #Close, 13 );
b1 := TroughBar( b2, #Close, 13 );
b4 := b2;
p1 := PriceClose( b1 );
p2 := PriceClose( b2 );
p3 := PriceClose( b3 );
p4 := p1 - ( p2 - p1 );
DrawDiamond( b1, p1, b2, p2, b3, p3, b4, p4, 0, #Gray, #Thick, #Silver,
true );

```

3.8 DrawEllipse

DrawEllipse(Bar1: integer; Price1: float; Bar2: integer; Price2: float; Pane: integer; Color: integer; Style: integer; FillColor: integer; BehindPrices: boolean);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Draws an ellipse, the corners of which are specified by the parameters *Bar1/Price1* and *Bar2/Price2*.

<i>Pane</i>	Specifies which on which pane to draw the ellipse. Pass 0 for the Price Pane, 1 for the Volume Pane, or use the return value of a CreatePane call for a custom pane.
<i>Color</i>	Controls the color used to draw the ellipse.
<i>Style</i>	Controls the type of line used. You can use the constants #Thin , #Thick or #Dotted for <i>Style</i> .
<i>FillColor</i>	Specifies the color with which to fill the ellipse. Pass -1 to draw an unfilled ellipse.
<i>BehindPrices</i>	Controls whether the ellipse is drawn behind or in front of the price bars.

Example

```

var BAR, PRICE: integer;
Bar := TroughBar( BarCount - 1, #Low, 13 );
Price := PriceLow( Bar );
DrawEllipse( Bar - 4, Price * 1.02, Bar + 4, Price * 0.98, 0, #RedBkg,
#Thin, #RedBkg, true );
Bar := PeakBar( BarCount - 1, #High, 13 );
Price := PriceHigh( Bar );
DrawEllipse( Bar - 4, Price * 1.02, Bar + 4, Price * 0.98, 0,
#GreenBkg, #Thin, #GreenBkg, true );

```

3.9 DrawHorzLine

DrawHorzLine(Value: float; Pane: integer; Color: integer; Style: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Draws a horizontal line on a *Pane*, at the value specified in the *Value* parameter.

Example

```

{ Draw a line at the psychologically important 1000 level }
DrawHorzLine( 1000, 0, #Green, #Dotted );

```

3.10 DrawImage

DrawImage(Bitmap: string; Pane: integer; Bar: integer; Price: float; TopDown: boolean);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Draws a bitmap image on the chart in the specified *Pane*. The image is drawn with a transparent area. The transparent color is determined by the color of the bitmap's lower left pixel.

<i>Bitmap</i>	The <i>Bitmap</i> parameter must contain the name of a bitmap file (bmp) that resides in the "Bitmaps" folder directly under the main Wealth-Lab Developer 4.0 folder. Provide the file name only, no path or file extension.
<i>Pane</i>	Specifies which on which pane to draw the ellipse. Pass 0 for the Price Pane, 1 for the Volume Pane, or use the return value of a CreatePane call for a custom pane.
<i>Bar</i>	Specifies the bar at which the image should be drawn. The image will be centered around the middle of the specified <i>Bar</i> .
<i>Price</i>	Indicates which price level (or indicator value, for non-price panes) at which to draw the image. (See <i>TopDown</i> next.)
<i>TopDown</i>	If the <i>TopDown</i> parameter is true, the top of the image will be placed at the specified <i>Price</i> level, otherwise the bottom of the image is placed at this level.

Example

```
var BAR: integer;
Bar := BarCount - 40;
DrawImage( 'UpArrow', 0, Bar, PriceLow( Bar ) * 0.995, true );
Bar := BarCount - 30;
DrawImage( 'DownArrow', 0, Bar, PriceHigh( Bar ) * 1.005, false );
```

3.11 DrawLabel

DrawLabel(Text: string; Pane: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Draws a *Text* label in the upper left corner of the specified *Pane*. You can call this function multiple times and the labels will be drawn one below the other. For more control over drawing text, use the **DrawText** function.

Example

```
{ Plot a 200 day moving average, and add a label to the chart }
PlotSeries( SMAseries( #Close, 200 ), 0, #Green, #Thin );
DrawLabel( '200 day SMA', 0 );
```

3.12 DrawLine

```
DrawLine( Bar1: integer; Price1: float; Bar2: integer; Price2: float; Pane: integer; Color: integer; Style:
integer );
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Draws a line between the two specified points. You specify a point as a bar/price pair. The *Bar1/Bar2* parameters correspond to the x-axis values, and *Price1/Price2* parameters to the y-axis values. The function automatically converts the bar/price pairs into drawing coordinates on the chart so you can more easily establish points for your lines.

Example

```
{ Draw a line between the last 2 peaks }
var P1, P2: float;
var BAR, PB1, PB2: integer;
Bar := BarCount - 1;
p1 := Peak( Bar, #High, 4 );
pb1 := PeakBar( Bar, #High, 4 );
p2 := Peak( pb1, #High, 4 );
pb2 := PeakBar( pb1, #High, 4 );
DrawLine( pb1, p1, pb2, p2, 0, #Red, #Dotted );
```

3.13 DrawRectangle

```
DrawRectangle( Bar1: integer; Price1: float; Bar2: integer; Price2: float; Pane: integer; Color: integer;
Style: integer; FillColor: integer; BehindPrices: boolean );
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Draws a rectangle, the corners of which are specified by the parameters *Bar1/Price1* and *Bar2/Price2*.

- | | |
|---------------------|---|
| <i>Pane</i> | Specifies which on which pane to draw the rectangle. Pass 0 for the Price Pane, 1 for the Volume Pane, or use the return value of a CreatePane call for a custom pane. |
| <i>Color</i> | Controls the color used to draw the rectangle. |
| <i>Style</i> | Controls the type of line used. You can use the constants #Thin , #Thick or #Dotted for <i>Style</i> . |
| <i>FillColor</i> | Specifies the color to fill the rectangle with. Pass -1 to draw an unfilled rectangle. |
| <i>BehindPrices</i> | Controls whether the rectangle is drawn behind the price bars or in front of them. |

Example

```
var P1, P2: float;
var BAR, B1, B2, P2: integer;
Bar := BarCount - 1;
b1 := PeakBar( Bar, #Close, 10 );
b2 := TroughBar( Bar, #Close, 10 );
p1 := PriceClose( b1 );
p2 := PriceClose( b2 );
DrawRectangle( b1, p1, b2, p2, 0, #Blue, #Thick, #BlueBkg, true );
```

3.14 DrawRoundRect

DrawRoundRect(Bar1: integer; Price1: float; Bar2: integer; Price2: float; Pane:integer; Color: integer; Style: integer; FillColor: integer; BehindPrices: boolean);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Draws a rectangle with rounded corners, which are specified by the parameters *Bar1/Price1* and *Bar2/Price2*.

<i>Pane</i>	Specifies which on which pane to draw the rectangle. Pass 0 for the Price Pane, 1 for the Volume Pane, or use the return value of a CreatePane call for a custom pane.
<i>Color</i>	Controls the color used to draw the rectangle.
<i>Style</i>	Controls the type of line used. You can use the constants #Thin , #Thick or #Dotted for <i>Style</i> .
<i>FillColor</i>	Specifies the color to fill the rectangle with. Pass -1 to draw an unfilled rectangle.
<i>BehindPrices</i>	Controls whether the rectangle is drawn behind the price bars or in front of them.

Example

```
var P1, P2: float;
var B1, B2: integer;
b1 := BarCount - 50;
b2 := BarCount - 10;
p1 := Highest( b2, #High, 40 );
p2 := Lowest( b2, #Low, 40 );
DrawRoundRect( b1, p1, b2, p2, 0, #Navy, #Thick, -1, false );
```

3.15 DrawText

DrawText(Text: string; Pane: integer; x: integer; y: integer; Color: integer; Size: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Writes the specified *Text* to a *Pane*. The *X* and *Y* parameters control the placement of the text, expressed as pixels from the upper left corner of the pane. *Color* refers to font color, and *Size* to font size. Standard font size is 8.

Example

```
{ Draw the result of our commentary to the volume pane }
var COMMENTARYSTRING: string;
CommentaryString := 'This is my advice, now listen closely ...';
DrawText( CommentaryString, 1, 4, 4, #Black, 8 );
```

3.16 DrawTriangle

DrawTriangle(Bar1: integer; Price1: float; Bar2: integer; Price2: float; Bar3: integer; Price3: float; Pane: integer; Color: integer; Style: integer; FillColor: integer; BehindPrices: boolean);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Draws a triangle, the three corners of which are specified by the parameters *Bar1/Price1*, *Bar2/Price2*, and *Bar3/Price3*.

<i>Pane</i>	Specifies which on which pane to draw the triangle. Pass 0 for the Price Pane, 1 for the Volume Pane, or use the return value of a CreatePane call for a custom pane.
<i>Color</i>	Controls the color used to draw the triangle.
<i>Style</i>	Controls the type of line used. You can use the constants #Thin , #Thick or #Dotted for <i>Style</i> .
<i>FillColor</i>	Specifies the color with which to fill the triangle. Pass -1 to draw an unfilled diamond.
<i>BehindPrices</i>	Controls whether the triangle is drawn behind or in front of the price bars.

Example

```
var PRICE1, PRICE2, PRICE3: float;
var BAR2, BAR, BAR1, BAR3: integer;
Bar := BarCount - 1;
Bar2 := PeakBar( Bar, #High, 15 );
Bar1 := TroughBar( Bar2, #Low, 15 );
Bar3 := Bar2 + ( Bar2 - Bar1 );
Price1 := PriceLow( Bar1 );
Price2 := PriceHigh( Bar2 );
Price3 := Price1;
DrawTriangle( Bar1, Price1, Bar2, Price2, Bar3, Price3, 0, #Olive,
#Thick, -1, false );
```

3.17 EnableNotes

EnableNotes(Enable: boolean);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Turns on (*Enable* = true) or off the display of notes above and below trades in the chart, such as "Buy 500 @10.00". If a trading system generates a multitude of trades, turning this option off will result in a much less cluttered chart.

Remarks

- See also: **EnableTradeNotes**

Example

```
{ Turn off those pesky notes if there are too many trades }
if PositionCount > 20 then
  EnableNotes( false );
```

3.18 EnableTradeNotes

EnableTradeNotes(Text: boolean; Arrow: boolean; Circle: boolean);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Turns on or off the display of textual and graphical notes above and below trades in the chart. If a Trading System generates a multitude of trades, passing **false** for the *Text*, *Arrow*, and/or *Circle* parameters will result in a much less cluttered chart.

<i>Text</i>	Controls whether or not textual notes are drawn on the chart, such as "Buy 200 @5.00".
<i>Arrow</i>	Controls whether or not buy and sell arrows appear above/below the bar where trades are opened and closed.
<i>Circle</i>	Controls whether the circles are drawn at the exact spot where trades occur on the bar. Additionally, if <i>Circle</i> is false, the horizontal dotted line that is normally drawn for open trades is not drawn.

Remarks

- This function supersedes the original **EnableNotes** function, which allowed the text notes only to be turned off.

Example

```
{ Turn off those pesky notes if there are many trades, show arrows only
}
if PositionCount > 20 then
  EnableTradeNotes( false, true, false );
```

3.19 HidePaneLines

HidePaneLines;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Hides the black lines that are drawn between chart panes.

Example

```
HidePaneLines;
```

3.20 HideVolume

HideVolume;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Renders the volume pane invisible, providing more room to the Prices Pane in the chart.

Example

```
HideVolume;
```

3.21 PlotSeries

PlotSeries(Series: integer; Pane: integer; Color: integer; Style: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Plots the specified price *Series* on one of the panes of the chart.

<i>Series</i>	An integer Price Series handle or a WealthScript function that returns an [integer] Price Series handle.
<i>Pane</i>	Controls in which pane the series will be plotted. Pass 0 for the Price Pane, 1 for the Volume Pane, or use the return value of a CreatePane call for a custom pane.
<i>Color</i>	A 3-digit integer that specifies the RGB color, or one of the standard color constants, e.g. #Black , #Red , #Blue , etc.
<i>Style</i>	One of the following plot formatting constants: #Thin , #Dotted , #Thick , #Histogram , #ThickHist , or #Dots

Remarks

- Although the OHLC/V values are automatically rendered according to the selected chart style, you may also wish to plot the average or average-close Standard Price Series using the constants **#Average** or **#AverageC**, respectively.
- If your futures data contains Open Interest, create a new pane and pass the **#OpenInterest** constant for *Series*. See the **OpenInterest** example in the Data Access category of functions.

Example

```
{ Plot the CMO in a new Pane }
var NEWPANE, CMOSER: integer;
NewPane := CreatePane( 80, true, true );
CmoSer := CMOSeries( #Close, 20 );
PlotSeries( CmoSer, NewPane, #Blue, #Thick );
```

3.22 PlotSeriesLabel

PlotSeriesLabel(Series: integer; Pane: integer; Color: integer; Style: integer; Label: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Plots the specified Price *Series* on one of the *Panes* of the chart, and adds a descriptive *Label* to the chart in the same color as that used to plot the series.

<i>Series</i>	An integer Price Series handle or a function that returns an [integer] Price Series handle.
<i>Pane</i>	Controls in which pane the series will be plotted. Pass 0 for the Price Pane, 1 for the Volume Pane, or use the return value of a CreatePane call for a custom pane.
<i>Color</i>	A 3-digit integer that specifies the RGB color, or one of the standard color constants, e.g. #Black , #Red , #Blue , etc.
<i>Style</i>	One of the following plot formatting constants: #Thin , #Dotted , #Thick , #Histogram , #ThickHist , or #Dots

Label A string literal or variable used to describe the plotted *Series*.

Remarks

- The label is drawn only if **Plot Labels on Chart** is checked in **Tools|Options|Colors/Style**.
- See **PlotSeries** for additional remarks.

Example

```
{ Plot the CMO in a new Pane }
var NEWPANE, CMOSER: integer;
NewPane := CreatePane( 80, true, true );
CmoSer := CMOSeries( #Close, 20 );
PlotSeriesLabel( CmoSer, NewPane, #Blue, #Thick, 'CMO(Close,20)' );
```

3.23 PlotStops

PlotStops;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Causes the various stop levels and profit targets to be visually depicted as small dots on the chart. The various levels are drawn using the following colors:

Profit Target	Green
Stop Loss	Red
Breakeven	Blue
Trailing	Pink
Manual Stop	Brown

Remarks

- Stops are plotted for exit signals only on the bar for which they are active.
- Manual stops are plotted at the stop or limit price passed to the exit signal. For example, a stop is plotted at Bar by the **SellAtStop** *StopPrice* parameter, whereas a profit target is plotted by the *LimitPrice* parameter in **SellAtLimit**.
- Other signals that fall into the manual-stop/limit category include **SellAtTrailingStop**, **CoverAtLimit**, **CoverAtStop**, and **CoverAtTrailingStop**
- **PlotStops** *enables* the display of stops and should be called prior to the main trading loop. "Installed" AutoStops are actually processed by the **ApplyAutoStops** function.

See Also: The QuickRef entry for the **Min** function provides a more dynamic example with manual stops.

Example

```
var Bar: integer;
InstallProfitTarget( 10 );
InstallStopLoss( 5 );
InstallTrailingStop( 1, 50 );
PlotStops;
for Bar := 20 to BarCount - 1 do
begin
    ApplyAutoStops( Bar );
```



```

{ Arbitrarily short every 100 bars }
  if Bar Mod 100 = 0 then
    ShortAtMarket( Bar + 1, '' );
  end;

```

3.24 PlotSymbol

PlotSymbol(Symbol: string; Pane: integer; Color: integer; Style: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Plots the symbol specified in the *Symbol* parameter to the chart pane specified in the *Pane* parameter. If *Symbol* does not exist in one of your DataSources, or it does not contain data within the range of the Primary Series specified in the Data Loading control, a run-time error will be generated.

<i>Symbol</i>	A string literal or variable containing the desired symbol. You should use a symbol other than the one for the Primary Series (the current chart symbol).
<i>Pane</i>	Controls in which pane the series will be plotted. Pass 0 for the Price Pane, 1 for the Volume Pane, or use the return value of a CreatePane call for a custom pane.
<i>Color</i>	A 3-digit integer that specifies the RGB color, or one of the standard color constants, e.g. #Black , #Red , #Blue , etc.
<i>Style</i>	Controls how the symbol appears, and can be one of the following constants: #OHLC , #Candle , or #Line

Example

```

var nP: integer;
PlotSymbol( 'MSFT', 0, #Silver, #Candle );
nP := CreatePane( 100, true, true );
PlotSymbol( 'BORL', nP, #Blue, #OHLC );

```

3.25 PlotSyntheticSymbol

PlotSyntheticSymbol(Symbol: string; Open: integer; High: integer; Low: integer; Close: integer; Pane: integer; Color: integer; Style: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Creates a synthetic "symbol" that contains open, high, low and closing prices. The *Symbol* parameter specifies the identity of the synthetic symbol. The next four parameters should contain Price Series handles that contain the *Open*, *High*, *Low*, and *Close* Price Series, respectively.

<i>Symbol</i>	A string literal or variable expression containing the synthetic symbol's name.
<i>Open</i>	An integer Price Series handle or a WealthScript function that returns an [integer] Price Series handle to be used for the opening Price Series of the synthetic symbol.
<i>High</i> , <i>Low</i> , and <i>Close</i>	same as <i>Open</i> for the respective Price Series.

<i>Pane</i>	Controls in which pane the series will be plotted. Pass 0 for the Price Pane, 1 for the Volume Pane, or use the return value of a CreatePane call for a custom pane.
<i>Color</i>	A 3-digit integer that specifies the RGB color, or one of the standard color constants, e.g. #Black , #Red , #Blue , etc. Synthetic symbols are plotted as a single color and do not follow up/down bar coloring.
<i>Style</i>	Controls how the symbol appears, and can be one of the following constants: #OHLC , #Candle , or #Line , the latter of which plots only the <i>Close</i> series.

Note: It's possible (and quite likely) for candles/bars to sometimes appear "incorrect" for a synthetic symbol. This is due, for example, to the low prices not always being less than the open, high and close for certain bars. Consequently, these candle values do not always form into traditionally correct candles.

Example

```
var SMAPANE, O, H, L, C: integer;
SMAPane := CreatePane( 100, true, true );
O := SMAseries( #Open, 20 );
H := SMAseries( #High, 20 );
L := SMAseries( #Low, 20 );
C := SMAseries( #Close, 20 );
PlotSyntheticSymbol( 'SMACandle', O, H, L, C, SMAPane, #Blue, #Candle );
DrawLabel( 'SMA Candle', SMAPane );
```

3.26 SetBackgroundColor

SetBackgroundColor(Bar: integer; Color: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Lets you color the background area of all chart panes with a specified *Color*, on a bar-by-bar basis. You can use this feature to highlight bull and bear trends in the chart. When coloring the chart background, be sure to use light, pastel colors for the *Color* parameter. Good colors to use for backgrounds include 988 (**#RedBkg**), 898 (**#GreenBkg**), and 889 (**#BlueBkg**).

Remarks

- To individually set the background color of any pane, use **SetPaneBackgroundColor**.

Example

```
{ Identify bull and bear trends using the 52 week moving average }
var SMAWEEKLY, SMAWEEKLYDAILY, BAR: integer;
SetScaleWeekly;
SmaWeekly := SMAseries( #Close, 52 );
RestorePrimarySeries;
SmaWeeklyDaily := DailyFromWeekly( SmaWeekly );
for Bar := 52 to BarCount - 1 do
begin
  if PriceClose( Bar ) > GetSeriesValue( Bar, SmaWeeklyDaily ) then
    SetBackgroundColor( Bar, #GreenBkg )
  else
```

```

        SetBackgroundColor( Bar, #RedBkg );
    end;

```

3.27 SetBarColor

```
SetBarColor( Bar: integer; Color: integer );
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Lets you color a specific *Bar* of the chart's primary Price Series. Generally, you'll use **Tools|Options|Colors/Style** to set your default choices for Up/Down bar coloring, however, you may want to color bars based on an indicator value, for example.

Remarks

- **SetBarColor** has priority over the **SetBarColors** function.
- See **SetSeriesBarColor** to set the color of individual bars in a Price Series *other than* the primary Price Series.

Example

```

{ Color bars green when RSI < 20, otherwise
  color up days blue and down days red }
var Bar, hRSI, RSIPane: integer;

hRSI := RSISeries( #Close, 14 );
for Bar := 20 to BarCount - 1 do
begin
    if @hRSI[ Bar ] < 60 then
        SetBarColor( Bar, #Green )
    else
        if PriceClose( Bar ) > PriceClose( Bar - 1 ) then
            SetBarColor( Bar, #Blue )
        else
            SetBarColor( Bar, #Red );
    end;
RSIPane := CreatePane( 75, true, true );
PlotSeries( hRSI, RSIPane, #Blue, #Thick );

```

3.28 SetBarColors

```
SetBarColors( UpBars: integer; DownBars: integer );
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Specifies the bar colors for all bars on the chart. Specify the color for up bars (close > open) using the *UpBars* parameter, and down bars (close < open) with the *DownBars* parameter. This function is useful when sharing ChartScripts and you need to show a specific color scheme for your methodology.

Note: Use **Tools|Options|Colors/Style** to set your default choices for Up/Down bar coloring. See Also: **SetColorScheme**.

Example

```
SetBarColors( #Navy, #Maroon );
```

3.29 SetColorScheme

SetColorScheme(UpBars: integer; DownBars: integer; Volume: integer; Background: integer; GridLines: integer; MarginArea: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Lets you control the color complete scheme of the chart in a single statement. Specify the color of up bars, down bars, volume bars, background, gridlines and bottom margin area. This function is especially useful when sharing ChartScripts and you need to show a specific color scheme for your methodology.

Example

```
{ A slick black chart style }
SetColorScheme( #Lime, 922, #Olive, 001, 021, #Silver );
```

3.30 SetLogScale

SetLogScale(Pane: integer; UseLogScale: boolean);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Controls whether or not the specified *Pane* will use a semi-log scale as controlled by the *UseLogScale* parameter. A semi-log scale gives equal weight to percentage changes, rather than absolute value changes. For example, the distance from 1 to 10 will be the same size on the chart as the distance from 10 to 100. It's called "semi-log" because only the y-axis uses the log scale, whereas the x-axis [typically] remains evenly-spaced.

Example

```
SetLogScale( 0, true );
```

3.31 SetPaneBackgroundColor

SetPaneBackgroundColor(Bar: integer; Pane: integer; Color: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Sets the background color of the specified *Pane* to the color provided in the *Color* parameter, at the specified *Bar*. Specify a Pane parameter of zero for the price pane, one for the volume pane, or use custom pane created by the **CreatePane** function.

Remarks

- **SetPaneBackgroundColor** overrides **SetBackgroundColor** for the specified *Pane*.

Example

```
{ Plot RSI and CMO, color backgrounds to show overbought/oversold
levels }
var Bar: integer;
var RSIPane: integer;
RSIPane := CreatePane( 75, true, true );
PlotSeries( RSISeries( #Close, 14 ), RSIPane, 005, #Thick );
DrawLabel( 'RSI(Close,14)', RSIPane );
```

```

var CMOPane: integer;
CMOPane := CreatePane( 80, true, true );
PlotSeries( CMOSeries( #Close, 14 ), CMOPane, 009, #Thick );
DrawLabel( 'CMO(Close,14)', CMOPane );
for Bar := 20 to BarCount - 1 do
begin
  if RSI( Bar, #Close, 14 ) < 30 then
    SetPaneBackgroundColor( Bar, RSIPane, #GreenBkg )
  else if RSI( Bar, #Close, 14 ) > 70 then
    SetPaneBackgroundColor( Bar, RSIPane, #RedBkg );
  if CMO( Bar, #Close, 14 ) < -50 then
    SetPaneBackgroundColor( Bar, CMOPane, #GreenBkg )
  else if CMO( Bar, #Close, 14 ) > 50 then
    SetPaneBackgroundColor( Bar, CMOPane, #RedBkg );
end;

```

3.32 SetPaneMinMax

SetPaneMinMax(Pane: integer; Min: float; Max: float);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Applies minimum values to the low and high end of the selected *Pane*. Normally a pane's min and max values are dependent on the Price Series that are plotted within the pane. The following example uses **SetPaneMinMax** to make sure that the **RSI** overbought and oversold levels appear clearly in the pane.

Note: Wealth-Lab's charting engine will still automatically scale panes to plot values outside of the *Min* and *Max* values specified in **SetPaneMinMax**.

Example

```

{ Make sure full price range is always visible in the pane }
var RSIPANE: integer;
RSIPane := CreatePane( 100, true, true );
PlotSeries( RSISeries( #Close, 30 ), RSIPane, #Navy, #Thin );
SetPaneMinMax( RSIPane, 20, 80 );

```

3.33 SetSeriesBarColor

SetSeriesBarColor(Bar: integer; Series: integer; Color: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Lets you color individual bars of any *Series* that is plotted. For example, you can color bars of an oversold oscillator green and overbought red.

Remarks

- To prevent drawing specific Bars of an indicator, pass -1 as the *Color* parameter.

Example

```

{ Color Bars of the indicator based on oversold/overbought levels }
var RSISER, BAR, RSIPANE: integer;
RSISer := RSISeries( #Close, 30 );
for Bar := 0 to BarCount - 1 do

```


```
if RSI( Bar, #Close, 30 ) > 60 then
  SetSeriesBarColor( Bar, RSISer, #Red )
else if RSI( Bar, #Close, 30 ) < 40 then
  SetSeriesBarColor( Bar, RSISer, #Blue );
RSIPane := CreatePane( 100, true, true );
PlotSeries( RSISer, RSIPane, #Navy, #Thick );
```

4 CommissionScript Functions

4.1 Overview

You should always include real-world trading costs to add fidelity to your backtesting. The Option Dialog (F12) includes a "Trading Costs/Control" tab that provides selections for commissions and slippage that you will experience in real-world trading.

If your broker uses a flat-fee commission for each trade, then you may select the "per Trade" One Way Commission option, which simply deducts a fixed amount from each trade in a simulation. Likewise, the "per Share" option reduces a trade's gross profit or loss by the number of shares multiplied by the value entered. Still, these simple commission options do not include other small adjustments that your broker can make on a per trade basis, such as the SEC fee for sale transactions in the U.S., which at the time of this writing is \$0.0468 per \$1,000.

Some brokers use graduated commission schedules or base their fees on a percentage of trade volume. CommissionScripts give you complete control over calculating simple to the most complex commission schedules used by brokers worldwide. Using the special "CM" variables provided, you can emulate the your broker's calculation and assign the result to the **CMResult** variable. Once complete, save the script to the  **CommissionScripts** ChartScript folder. At this point, the script will be available as a selection in the CommissionScript drop down control in the Options Dialog.

For each trade processed during a simulation - both entries and exits - Wealth-Lab will execute the selected CommissionScript. The value calculated and applied to the CMResult variable will then be used as the trade's commission cost.

Note: If you find that no commissions are ever deducted when using your commission script, check the script for errors.

4.2 CMShares

CMShares(): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the number of shares (or contracts) for the commission calculation.

4.3 CMPrice

CMPrice(): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns either the entry or exit price, as required, of the position for the commission calculation.

4.4 CMEntry

CMEntry(): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns boolean *true* if the trade being processed is an entry signal. Consequently, CMEntry will be *false* when exiting a position. You can use this function to add the SEC fee for sales transactions in the U.S., for example.

Example:

```
{ My broker charges 10.99 per trade, but also adds the SEC sales fee }
var SECFee: float;
const SECRate = 0.0234;      // per $1,000

CMResult := 10.99;
if not CMEntry then
begin
    SECFee := SECRate * CMShares * CMPrice / 1000;
    SECFee := Round( SECFee * 100 ) / 100;
    CMResult := CMResult + SECFee;
end;
```

4.5 CMOrderType

CMOrderType(): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns an integer indicating the type of order used.

0 = Market
 1 = Stop
 2 = Limit
 3 = Close

Example:

```
{ My broker charges $9.99 for market orders, $11.00 for limit or stop
orders, and $12.50 to work an order at the close }
var Comish: float;

case CMOrderType of
0:
    Comish := 9.99;
1, 2:
    Comish := 11.00;
3:
    Comish := 12.50;
else
    { This one's on the house! }
    Comish := 0.0;
end;
CMResult := Comish;
```


4.6 CMSymbol

CMSymbol(): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the security symbol of the trade to which commissions will be applied.

4.7 CMDataSource

CMDataSource(): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the name of the DataSource to which **CMSymbol** belongs. You can test the DataSource name to use a different commission structure based on a DataSource.

Remarks

- In real time **CMDataSource** returns a blank string.

4.8 CMResult

CMResult(): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

You must assign the final result of the CommissionScript calculation to the special **CMResult** variable. **CMResult** is akin to the *Result* variable used to return the final result of a user-defined function in WealthScript. **CMResult** need not be declared and can be used as a normal float-type variable throughout the CommissionScript's process.

Example

```
{ Emulate commissions for a broker with the following fee structure:  
  1¢ for first 500 shares, ½¢ per share thereafter, and $1 minimum }  
if CMShares <= 500 then  
  CMResult := CMShares * 0.01  
else  
  CMResult := ( 500 * 0.01 ) + ( ( CMShares - 500 ) * 0.005 );  
  
if CMResult < 1 then  
  CMResult := 1;
```

5 Data Access Functions

5.1 Overview

The Data Access functions provide the methods to access the data from all the raw Price Series in the Primary Data Series (the symbol selected for the script) as well as other Standard Price Series information. Additionally, futures symbols' point, margin, and tick entries in the Future Symbol Manager can be easily accessed from within your script.

5.2 BarCount

BarCount: integer;

ChartScripts *SimuScripts PerfScripts CMScripts

Description

Returns the total number of bars available in the current chart.

Remarks

The first bar of any chart is Bar Number 0, and the last bar can be found by the expression `BarCount - 1`.

Example

```
{ A typical trading system main loop }
var Bar: integer;
for Bar := 30 to BarCount - 1 do
begin
  { ... Trading Rules ... }
end;
```

5.3 GetDate

GetDate(Bar: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the date of the specified *Bar* number. WealthScript represents date values as large integers - the year followed by month followed by day. For example, 1/23/2001 would be represented as 20010123. You can thus compare dates by simply using the standard arithmetic operators.

Example

```
{ Print the most recent date on the chart to the debug window }
var S: string;
s := IntToStr( GetDate( BarCount - 1 ) );
Print( s );
```

5.4 GetMargin

GetMargin: float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Margin for the current symbol being charted. Margin is the amount of funds required to maintain a single futures contract position. If the current symbol is not a futures symbol, the function returns zero. You define Margin in the Futures Symbols Manager, **Tools|Futures Symbols Manager (Ctrl+Alt+F)**.

Example

```
var Bar: integer;

{ Take on $20,000 margin per position }
if GetMargin > 0 then
  SetShareSize( Round( 20000 / GetMargin ) );

for Bar := 4 to BarCount() - 1 do
begin
  if LastPositionActive then
    SellAtStop( Bar + 1, Lowest( Bar, #Low, 3 ), LastPosition, '' )
  else
    BuyAtStop( Bar + 1, Highest( Bar, #High, 3 ), '' );
end;
```

5.5 GetPointValue

GetPointValue: float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Point Value for the current futures symbol being charted. The Point Value is the amount of profit made when prices increase a single point. If the current symbol is not a futures symbol, the function returns 1. You define Point Value in the Futures Symbols Manager, **Tools|Futures Symbols Manager (Ctrl+Alt+F)**.

Example

```
var PtValue: float;
var s: string;
PtValue := GetPointValue;
s := FormatFloat( '0.00', PtValue );
ShowMessage( 'The Point Value is ' + s );
```

5.6 GetSecurityName

GetSecurityName: string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the security name (company name) of the symbol currently being operated on. Not all DataSources provide the security name in which case a blank string is returned.

Example

```
ShowMessage( 'We're now running ' + GetSecurityName );
```

5.7 GetSymbol

GetSymbol: string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the symbol of the current chart.

Example

```
{ Show the closing price with the symbol in a chart label }
var X: float;
x := PriceClose( BarCount - 1 );
DrawLabel( 'Closing price for ' + GetSymbol + ' was '
          + FormatFloat( '##,##0.00', x ), 0 );
```

5.8 GetTick

GetTick: float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Tick value for the current primary symbol. The Tick value is the smallest incremental price move that a futures contract can make. You define Tick values in the Futures Symbols Manager, **Tools|Futures Symbols Manager (Ctrl+Alt+F)**.

Remarks

- In a SimuScript, **GetTick** returns the Tick value for the current futures symbol being position-sized.
- **GetTick** returns 0 for any symbol that does not have an entry in the Futures Symbols Manager, i.e., a stock. With Stock Mode selected in the DataSources main menu, **GetTick** always returns 0.

Example

```
{ The 89/13 Futures Breakout System }
var Tick, XLOW, XHIGH: float;
var BAR: integer;

Tick := GetTick;

for Bar := 90 to BarCount - 1 do
begin
  if LastPositionActive then
  begin
    if PositionLong( LastPosition ) then
    begin
      xLow := Lowest( Bar, #Low, 13 ) - TICK;
      SellAtStop( Bar + 1, xLow, LastPosition, '' );
    end
  else
  begin
```

```

    xHigh := Highest( Bar, #High, 13 ) + TICK;
    CoverAtStop( Bar + 1, xHigh, LastPosition, '' );
end;
else
begin
    xHigh := Highest( Bar, #High, 89 ) + TICK;
    xLow := Lowest( Bar, #Low, 89 ) - TICK;
    if BuyAtStop( Bar + 1, xHigh, '' ) then
        SetPositionRiskStop( LastPosition, Lowest( Bar, #Low, 13 ) )
    else if ShortAtStop( Bar + 1, xLow, '' ) then
        SetPositionRiskStop( LastPosition, Highest( Bar, #High, 13 ) );
    end;
end;
PlotSeries( HighestSeries( #High, 89 ), 0, 777, #Thin );
PlotSeries( LowestSeries( #Low, 89 ), 0, 777, #Thin );
PlotSeries( HighestSeries( #High, 13 ), 0, 888, #Thin );
PlotSeries( LowestSeries( #Low, 13 ), 0, 888, #Thin );

```

5.9 GetTime

GetTime(Bar: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the time of the specified *Bar*. WealthScript represents time values as integers - hour (24 hour clock format) followed by minute. For example, 1:00 PM is represented as 1300. You can thus easily test for specific time values or test times against each other using the standard arithmetic operators.

Non-intraday DataSources will always return a value of zero for **GetTime**.

Example

```

{ Buy only after 12 noon }
var NOON: integer;
var BAR: integer;
Noon := 1200;
for Bar := 0 to BarCount - 1 do
begin
    if GetTime( Bar ) > Noon then
        SetBarColor( Bar, #Red );
    end;
end;

```

5.10 OpenInterest

OpenInterest(Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the open interest for the specified *Bar*. Open Interest is the number of contracts currently open for a given futures contract. **OpenInterest** is available only for Futures DataSources. Non-Futures DataSources will always contain zero values for **OpenInterest**.

Use the **OpenInterest** function to return the open interest value as of a specified *Bar*.

If you need to access the complete Open Interest Price Series handle, use the `#OpenInterest` built-in constant.

Interpretation

Open Interest can be used to gauge market liquidity, similar to Volume.

Example

```
var OEPANE: integer;
OEPane := CreatePane( 50, false, true );
SetPaneMinMax( OEPane, 0, 100 );
PlotSeries( #OpenInterest, OEPane, #Green, #Histogram );
DrawLabel( 'Open Interest', OEPane );
```

5.11 PriceAverage

PriceAverage(Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the average price for the specified *Bar*. Use PriceAverage to return the value of individual average prices at specific bars. The Average Price is defined as $(High + Low) / 2$. If you need to access the complete average price Price Series, use the `#Average` constant instead.

Example

```
{ Plot a moving average of daily average prices }
var X: float;
var BAR: integer;
x := PriceAverage( Bar );
PlotSeries( SMAseries( #Average, 30 ), 0, #Blue, #Thick );
```

5.12 PriceAverageC

PriceAverageC(Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the average price for the specified *Bar*. Use PriceAverage to return the value of individual average prices at specific bars. The Average Price is defined as $(High + Low + Close) / 3$. If you need to access the complete average price Price Series, use the `#AverageC` constant instead.

Example

```
{ Plot a moving average of close-weighted daily average prices }
var X: float;
var BAR: integer;
x := PriceAverageC( Bar );
PlotSeries( SMAseries( #AverageC, 30 ), 0, #Blue, #Thick );
```

5.13 PriceClose

PriceClose(Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the closing price for the specified *Bar*. Use PriceClose to return the value of individual closing prices at specific bars. If you need to access the complete closing price Price Series, use the **#Close** constant instead.

Example

```
{ Have we had an up day? }
var BAR: integer;
for Bar := 1 to BarCount - 1 do
  if PriceClose( Bar ) > PriceClose( Bar - 1 ) then
    SetBarColor( Bar, 853 );
```

5.14 PriceHigh

PriceHigh(Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the high price for the specified *Bar*. Use **PriceHigh** to return the value of intraday highs at specific bars. If you need to access the complete high price Price Series, use the **#High** constant instead.

Example

```
{ Have we achieved a new 200 bar high in the last 20 bars? }
var X, X2: float;
var BAR: integer;
for Bar := 200 to BarCount - 1 do
begin
  x := Highest( Bar, #High, 20 );
  x2 := Highest( Bar, #High, 200 );
  if x = x2 then
    SetBarColor( Bar, 083 );
end;
```

5.15 PriceLow

PriceLow(Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the low price for the specified *Bar*. Use PriceLow to return the value of intraday lows at specific bars. If you need to access the complete low price Price Series, use the **#Low** constant instead.

Example

```
{ Set a stop at the low of the entry bar }
var BAR: integer;
```

```

for Bar := 80 to BarCount - 1 do
begin
  if LastPositionActive then
    SellAtStop( Bar + 1, Lowest( Bar, #Low, 20 ), LastPosition, 'Stop'
)
  else
  begin
    if BuyAtStop( Bar + 1, Highest( Bar, #High, 80 ), 'Stop' ) then
      SetPositionRiskStop( LastPosition, Lowest( Bar, #Low, 20 ) );
    end;
  end;
end;

```

5.16 PriceOpen

PriceOpen(Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the opening price for the specified *Bar*. Use PriceOpen to return the value of opening prices at specific bars. If you need to access the complete open price PriceSeries, use the **#Open** constant instead.

Example

```

{ If we have a gap up open, buy it }
var BAR: integer;
for Bar := 1 to BarCount - 1 do
begin
  if LastPositionActive then
    SellAtMarket( Bar + 1, LastPosition, '1 Day' )
  else
  begin
    if PriceOpen( Bar ) > PriceHigh( Bar - 1 ) * 1.05 then
      BuyAtMarket( Bar, 'Gap' );
    end;
  end;
end;

```

5.17 Volume

Volume(Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the volume for the specified *Bar*. Use Volume to return the volume at specific bars. If you need to access the complete volume Price Series, use the **#Volume** constant instead.

Example

```

{ Plot a 30 day moving average of Volume }
var VOL_SMA: integer;
VolSMA := SMA_Series( #Volume, 30 );
PlotSeries( VolSMA, 1, #Red, #Thin );

```


6 Date/Time Functions

6.1 Overview

Generally speaking, the Date and Time category of WealthScript functions give you access to date/time-related information of an underlying Price Series at specific Bar Numbers. More general date/time information is available, such as the current date/time of your computer and important market events, like option expiry dates.

6.2 BarInterval

BarInterval: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the bar interval in minutes, seconds, or ticks (as required) for intraday charts. Returns zero in non-intraday charts.

Example

```
var bi: integer;
bi := BarInterval;
if bi < 1 then
  ShowMessage('Not an intraday chart')
else
  ShowMessage('The intraday bar interval is '
    + IntToStr( bi ) );
```

6.3 BarNum

BarNum(Bar: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Bar Number of the specified *Bar* within the trading day for intraday charts. Intraday charts can be minute, second, or tick-based. The first bar of the day has a **BarNum** of zero. Non-intraday charts always return zero for **BarNum**.

Example

```
{ Color the middle of the trading day }
var MAXBARS, BAR, PCT: integer;
{ First determine how many bars there are in one day }
MaxBars := 0;
for Bar := BarCount - 1 downto 1 do
  if BarNum( Bar ) = 0 then
    begin
      MaxBars := BarNum( Bar - 1 );
      Break;
    end;
if MaxBars = 0 then
  Exit;
{ Now color the bars 40 - 60% within the day's range }
for Bar := 0 to BarCount - 1 do
```

```

begin
  pct := BarNum( Bar ) / MaxBars;
  if ( pct >= 0.4 ) and ( pct <= 0.6 ) then
    SetBarColor( Bar, #Olive );
  end;

```

6.4 CurrentDate

CurrentDate: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the current system date in yyyyMMdd format.

Example

```

{ Draw the current date in yyyyMMdd integer format }
DrawLabel( IntToStr( CurrentDate ), 0 );

{ Use DateToStr to draw the current date using your Window's settings
in the volume pane }
DrawLabel( DateToStr( CurrentDate ), 1 );

```

6.5 CurrentTime

CurrentTime: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the current system time in hhmm format.

Example

```

{ Draw the current time in hhmm integer format }
DrawLabel( IntToStr( CurrentTime ), 0 );

{ Use TimeToStr to draw the current time using your Window's settings
in the volume pane }
DrawLabel( TimeToStr( CurrentTime ), 1 );

```

6.6 DateTimeToBar

DateTimeToBar(Date: integer; Time: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Bar Number on an *intraday* chart that corresponds to the specified *Date* and *Time* values. If there is no bar that corresponds to the specified date, the function returns -1.

Remarks

- *Date* is an integer value with the format yyyyMMdd, where yyyy is the year, MM is the two-digit month (01 through 12), and dd is the two-digit day (01 through 31, depending on the month).

- *Time* is an integer value with the format hhnn, where hh is the hour (0 through 23), and nn is the two-digit minute (00 through 59).
- If either *Date* or *Time* is not a valid, "in range" value, a run-time error will result.
- **DateTimeToBar** can be used to return a Bar Number on a non-intraday chart by passing zero as *Time*, though **DateToBar** is preferred.

Example

```
{ $I 'EnterAtPrice' }
{ "Load" a specific trade at 1050 on 20040123, try with AAPL }
var Bar: integer;

Bar := DateTimeToBar( 20040123, 1050 );
if Bar = -1 then
  Print( 'This bar does not exist in the chart' )
else
  if not EnterAtPrice( Bar, 22.65, 'Long', 'Buy' ) then
    Print( 'Buy failed on bar ' + IntToStr( Bar ) );
```

6.7 DateToBar

DateToBar(Date: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Bar Number on the chart that corresponds to the specified *Date* value. If there is no bar that corresponds to the specified date, the function returns -1.

Remarks

- *Date* is an integer value with the format yyyyMMdd, where yyyy is the year, MM is the two-digit month (01 through 12), and dd is the two-digit day (01 through 31, depending on the month).
- If *Date* is not a valid, "in range" value, a run-time error will result.
- **DateToBar** returns the Bar Number of the *first* bar of the specified *Date* in the intraday data.

Example

```
{ Highlight my birthday bars }
var Y, DT, B: integer;
for y := 1980 to 2020 do
begin
  dt := y * 10000 + 825;
  b := DateToBar( dt );
  if b >= 0 then
    SetBarColor( b, #Blue );
end;
```

6.8 DateToStr

DateToStr(Date: integer): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns a string representation of the specified integer *Date* value. The date representation of the resulting string is determined by your computer's Regional Options for "Short date format" on the Date tab.

Example

```
{ Record the dates where RSI was at an extremely high level }
var BAR: integer;
for Bar := 20 to BarCount - 1 do
  if RSI( Bar, #Close, 30 ) > 75 then
    Print( DateToStr( GetDate( Bar ) ) + ' '
      + FormatFloat( '##0.00', RSI( Bar, #Close, 30 ) ) );
```

6.9 DayOfWeek

DayOfWeek(Bar: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the day of the week of the specified *Bar* number. Sunday is the first day of the week and Saturday is the seventh. You can use the provided constants **#Monday** through **#Friday** to reference day of weeks in your WealthScript code.

Example

```
{ Color Mondays red and Fridays Green }
var BAR: integer;
for Bar := BarCount - 1 downto 1 do
begin
  if DayOfWeek( Bar ) = #MONDAY then
    SetBarColor( #Red )
  else if DayOfWeek( Bar ) = #FRIDAY then
    SetBarColor( #Green );
end;
```

6.10 DaysBetween

DaysBetween(Bar1: integer; Bar2: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Determines the number of Calendar Days (including weekends) between the two specified bars numbers, *Bar1* and *Bar2*.

Remarks

- **DaysBetween** returns a negative number of days if the date specified by *Bar2* occurs prior to the date of *Bar1*.
- See also: **DaysBetweenDates**

Example

```
var BAR, P, INBAR, OUTBAR, BARSINTRADE, DAYSINTRADE: integer;
InstallProfitTarget( 6 );
InstallStopLoss( 6 );
for Bar := 20 to BarCount - 1 do
```

```

begin
  ApplyAutoStops( Bar );
  if TurnUp( Bar, WMASeries( #Close, 20 ) ) then
    BuyAtMarket( Bar + 1, '' );
  end;
  if PositionCount > 0 then
  begin
    p := LastPosition;
    InBar := PositionEntryBar( p );
    if PositionActive( p ) then
      OutBar := BarCount - 1
    else
      OutBar := PositionExitBar( p );

    BarsInTrade := OutBar - InBar + 1;
    DaysInTrade := DaysBetween( InBar, OutBar ) + 1;
    DrawLabel( 'The last trade was for ' + IntToStr( BarsInTrade ) + '
Bars', 0 );
    DrawLabel( 'and ' + IntToStr( DaysInTrade ) + ' days', 0 );
  end;

```

6.11 DaysBetweenDates

DaysBetweenDates(Date1: integer; Date2: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Determines the number of calendar days including weekends between the two specified dates, *Date1* and *Date2*, in yyyyMMdd (standard) format.

Remarks

- **DaysBetweenDates** returns a negative number of days if *Date2* is chronologically prior to *Date1*.
- See also: **DaysBetween**

Example

```

{ Calculate the number of days between the last bar and the date
entered }
var days, date1, date2: integer;
date1 := GetDate( BarCount - 1 );
date2 := StrToInt( Input( 'Enter a yyyyMMdd date' ) );
days := DaysBetweenDates( date1, date2 );
ShowMessage( IntToStr( days ) + ' days between ' + DateToStr( date1 )
+ ' and ' + DateToStr( date2 ) );

```

6.12 GetDay

GetDay(Bar: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

GetDay returns the day of the month of the specified *Bar*.

Example

```

{ Highlight all Friday the 13ths }

```

```

var BAR: integer;
for Bar := 0 to BarCount - 1 do
begin
  if DayOfWeek( Bar ) = #Friday then
    if GetDay( Bar ) = 13 then
      begin
        SetBarColor( Bar, #Red );
        AnnotateBar( 'Look out!', Bar, true, #Red, 10 );
      end;
    end;
end;

```

6.13 GetHour

GetHour(Bar: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the hour of the specified *Bar*. Hours are expressed using a 24 hour clock, e.g., 13 indicates the bar falls between 1:00PM and 1:59PM, inclusive. Non-intraday charts will always return zero for **GetHour**.

Example

```

{ Buy on Breakout and close at 3:00 PM }
var Bar: integer;
for Bar := 20 to BarCount - 1 do
begin
  if LastPositionActive then
    begin
      if GetHour( Bar ) = 15 then
        SellAtMarket( Bar, LastPosition, '3:00' );
      end
    else if GetHour( Bar ) < 12 then
      BuyAtStop( Bar + 1, Highest( Bar, #High, 20 ), 'Buy Stop' );
    end;
end;

```

6.14 GetMinute

GetMinute(Bar: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the minute of the specified *Bar* (0 to 59). Non-intraday bars will always return zero for **GetMinute**.

Example

```

{ Color Intraday bars by minute }
var Bar, n: integer;
for Bar := 0 to BarCount - 1 do
begin
  n := ( GetMinute( Bar ) * 100 ) div 60;
  SetBarColor( Bar, n );
end;

```

6.15 GetMonth

GetMonth(Bar: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

GetMonth returns the month at the specified *Bar*. Month values range from 1 to 12 (January to December).

Example

```
{ We think February is an awful month for stocks }
var Bar: integer;
for Bar := 0 to BarCount - 1 do
begin
  if not LastPositionActive then
    if GetMonth( Bar ) = 2 then
      ShortAtMarket( Bar, 'Feb Blues' );
  if LastPositionActive then
    if GetMonth( Bar ) = 3 then
      CoverAtMarket( Bar, LastPosition, 'Exit Short' );
end;
```

6.16 GetYear

GetYear(Bar: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

GetYear returns the year at the specified *Bar*.

Example

```
{ Change our tactics for the new millenium }
var Bar: integer;
for Bar := 0 to BarCount - 1 do
begin
  if GetYear( Bar ) < 2000 then
  begin
    { .. old tactics .. }
  end
  else
  begin
    { .. new millenium tactics .. }
  end;
end;
```

6.17 IsLeapYear

IsLeapYear(Year: integer): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns boolean true if the specified calendar *Year* is a leap year. Returns false in all other cases.

Remarks

- **IsLeapYear**(*Year*) simply returns the same result as (*Year* **Mod** 4 = 0).

Example

```
var Bar: integer;

for Bar := 100 to BarCount - 1 do
begin
  if LastPositionActive then
  begin
    if CrossUnder( Bar, #Close, SMAseries( #Close, 100 ) ) then
      SellAtMarket( Bar + 1, LastPosition, '' );
    end
  else
  begin
    if not IsLeapYear( GetYear( Bar ) ) then // Take leap years off
      if CrossOver( Bar, #Close, SMAseries( #Close, 100 ) ) then
        BuyAtMarket( Bar + 1, '' );
      end;
    end;
  end;
end;
```

6.18 LastBar

LastBar(Bar: integer): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns boolean true if the specified *Bar* is the last bar of the day for intraday charts. Returns false in all other cases.

Remarks

- **LastBar** is most useful in backtesting to detect the last intraday bar, especially for incomplete market days or those sessions that close earlier than normal.
- For real-time charts, **LastBar** returns *true* for the final bar of the sessions that matches the "Market Closing Time" in the real-time data loading filter control, when activated. If the filter is not activated, **LastBar** is undefined and can return either *true* or *false* for the final bar in the chart.
- For real-time trading, consider using the **PortfolioSynch** function in combination with a specific test for the market closing time (adjust each day if required for short market sessions) using **GetTime**.

Example

```
{ Daytrading SMA crossover script (backtesting only) that closes all
positions at the end of the day. }
var Bar, p, hMASlow, hMAFast: integer;

hMAFast := SMAseries( #Close, 10 );
hMASlow := SMAseries( #Close, 30 );

for Bar := 30 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin { Entry Rules - don't enter on LastBar! }
    if not LastBar( Bar ) then
      if CrossOver( Bar, hMAFast, hMASlow ) then
        BuyAtMarket( Bar + 1, 'XOver' );
      end
    end;
  end;
end;
```



```

else { Exit Rules }
begin
  p := LastPosition;
  if LastBar( Bar ) then
    SellAtClose( Bar, p, 'EOD' )
  else
    begin { normal intraday exit logic }
      if CrossUnder( Bar, hMAFast, hMASlow ) then
        SellAtMarket( Bar + 1, p, 'XUnder' );
      end;
    end;
end;

PlotSeries( hMAFast, 0, #Green, #Thin );
PlotSeries( hMASlow, 0, #Red, #Thin );

```

6.19 OptionExpiryDate

OptionExpiryDate(Bar: integer): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns boolean true if the specified *Bar* falls on an option expiration date. The expiration date for all listed stock options in the U.S. is the third Friday of the expiration month.

Note: If the normal Friday expiration date falls on a holiday, the **OptionExpiryDate** is the preceding Thursday.

Example

```

{ Annotate Option Expiry Dates on the Chart }
var Bar: integer;
for Bar := 0 to BarCount - 1 do
  if OptionExpiryDate( Bar ) then
    begin
      DrawCircle( 4, 0, Bar, PriceOpen( Bar ), #Navy, #Thick );
      DrawCircle( 4, 0, Bar, PriceClose( Bar ), #Blue, #Thick );
    end;
end;

```

6.20 StrToDate

StrToDate(Value: string): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Converts the string parameter *Value* into an integer representing a WealthScript date value, which has a format of YYYYMMDD. The *Value* string must be in your computer's short date format, otherwise a run-time error will result.

Example

```

{ This was a bad day }
var dt, Bar: integer;
dt := StrToDate( '10/19/1987' );
Bar := DateToBar( dt );
if Bar > -1 then

```

```
SetBarColor( Bar, #Red );
```

6.21 StrToTime

StrToTime(Value: string): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Converts the string parameter *Value* into an integer representing a WealthScript time value, which is an integer having the format hhmm. The *Value* string must be a valid time format, e.g., '14:30', '2:30 PM', etc., otherwise a run-time error will result.

Example

```
{ Only take action after 2:00 PM }  
var Bar: integer;  
for Bar := 0 to BarCount - 1 do  
begin  
  { ... }  
  if GetTime( Bar ) > StrToTime( '2:00 PM' ) then  
  begin  
    end;  
  end;  
end;
```

6.22 TimeToStr

TimeToStr(Time: integer): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Converts the specified integer *Time* value into a string for display purposes. The time representation of the resulting string is determined by your computer's Regional Options for "Time format" on the Time tab.

Example

```
{ Draw the time of the last bar on the chart }  
DrawLabel( TimeToStr( GetTime( BarCount - 1 ) ), 0 );
```

7 File Access Functions

7.1 Overview

The File Access category of functions give you the ability to easily work with data from external ASCII text files or generate data from within your ChartScripts to export data for later review.

7.2 FileClear

```
FileClear( File: integer );
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Clears the contents of the file handle specified in the *File* parameter. Use this function if you've written lines to file and want to clear the existing contents and start fresh.

Note: File handles are returned by either the **FileOpen** or **FileCreate** functions.

7.3 FileClose

```
FileClose( File: integer );
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Explicitly closes the selected File and removes the resources allocated by the file. The File Handle represented by the *File* parameter is no longer valid following this call, and should not be used in subsequent File Access functions.

Note: File handles are returned by either the **FileOpen** or **FileCreate** functions.

Files are automatically closed after the script completes processing. During WatchList Scans or \$imulations, files are automatically closed after the complete Scan or \$imulation. Consequently, when opening a file using **FileCreate**, each symbol run during a Scan or \$imulation can append lines of data to a single output file without deleting the file that was created at the beginning of the Scan or \$imulation.

7.4 FileCreate

```
FileCreate( FileName: string ): integer;
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Creates a new operating system file with the specified *FileName*. A file handle is returned by the function call. Use this file handle in subsequent calls to **FileWrite**. If a file with the specified file name already exists, the file is deleted and a new one created in its place.

FileName A string representing the full path location and name for the new file. If a path is not specified, as in the example below, the file will be created in the main Wealth-Lab Developer 4.0 directory.

Example

```
{ Create a file to store analysis results }
var f: integer;
f := FileCreate( GetSymbol + '.txt' );
```

7.5 FileEOF

FileEOF(File: integer): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true if the specified *File* is currently at "end of file".

Example

```
{ Dump contents of Win.ini }
var F: integer;
f := FileOpen( 'c:\Windows\win.ini' );
while not FileEOF( f ) do
  Print( FileRead( f ) );
```

7.6 FileFlush

FileFlush(File: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Normally, data written to a file using **FileWrite** isn't physically written to the underlying operating system file until after the script completes. You can use **FileFlush** to cause the contents of *File*, the file handle, to be written to the operating system file immediately.

7.7 FileOpen

FileOpen(FileName: string): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Opens an existing operating system file with the specified *FileName*. The function returns a file handle that should be used in subsequent calls to **FileRead** or **FileWrite**. If the file does not exist, **FileOpen** will create it.

FileName A string representing the full path location and name of the file. If a path is not specified, as in the example below, the file will be assumed to exist in the main Wealth-Lab Developer 4.0 directory.

Example

```
{ Open a file to read external data for the symbol }
var f: integer;
```

```
f := FileOpen( GetSymbol + '.txt' );
```

7.8 FileRead

```
FileRead( File: integer ): string;
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Reads a line of data from the specified *File* handle. The *File* parameter should be a file handle that was returned by the **FileOpen** function. The function returns the next line of the file as a string. If there are no more lines in the file the function returns a blank string. Use the **FileEOF** function to test whether a file is truly at end of file.

Example

```
{ Read a line from the file into a string variable }
var s: string;
var fh: integer;
fh := FileOpen( 'c:\myfile.txt' );
s := FileRead( fh );
```

7.9 FileWrite

```
FileWrite( File: integer; Line: string );
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Writes a single *Line* of text to the specified *File* handle. You can write to new files that were created with **FileCreate** or to existing files that were opened with **FileOpen**. Read and write file operations maintain separate file pointers, so you can read from a file created with **FileOpen** and use **FileWrite** to write to the same File Handle without disrupting the read.

Example

```
{ Write an analysis file that consists of RSI level and price change 20
bars out }
var f, Bar: integer;
var s: string;
var x: float;
f := FileCreate( 'RSI Analysis.csv' );
for Bar := 20 to BarCount - 21 do
begin
  x := PriceClose( Bar + 20 ) - PriceClose( Bar );
  x := ( x / PriceClose( Bar ) ) * 100;
  s := FloatToStr( RSI( Bar, #Close, 20 ) ) + ',' + FloatToStr( x );
  FileWrite( f, s );
end;
```

8 Fundamental Data Access Functions

8.1 FundamentalPriceSeriesAverage

FundamentalPriceSeriesAverage(Item: string; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns a Price Series containing the rolling average of the fundamental data *Item* for the specified *Period*. This function is useful for creating a rolling average of any fundamental data item, particularly economic indicators.

Remarks

- **Wealth-Lab Pro only.** See the *Fundamental Data and Economic Indicator Definitions Guide* for valid *Item* parameters.
- If *Item* is not found, the function raises an error, which can be detected using a try/except/end block (see "Error Handling" in the WealthScript Language Guide).
- **FundamentalPriceSeriesAverage** returns **0** value for all bars if the full number of specified *Period* are not available, which is typical behavior at the beginning of the fundamental series.
- To access fundamental data of secondary symbols, call **SetPrimarySeries** first.

Example

```
var P, A, AV: integer;
P := CreatePane( 100, true, true );
A := FundamentalPriceSeries( 'assets' );
AV := FundamentalPriceSeriesAverage( 'assets', 4 );
PlotSeries( A, p, #Red, #ThickHist );
PlotSeries( AV, p, #Black, #Thick );
```

8.2 GetFundamentalDetail

GetFundamentalDetail(Bar: integer; Item: string; Detail: string): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Provides access to item-specific *Details* associated with the fundamental data *Item* that is synchronized to the specified *Bar* number. *Detail* is *Item*-dependent and can be any one of the strings on the left side of the "=" sign in the Data string. Use **FundamentalItemData** to access the entire Data string of an *Item*.

For example, consider the following data string:

```
FY=2006;CurQTR=3;CurQTRMonth=9;CalendarYear=2006;EPSCurFY=2.57;EPSNextFY=2.874;
```

For the Data string above, *Detail* can be any of the following strings: 'FY' or 'CurQTR' or 'CurQTRMonth' or 'CalendarYear' or 'EPSCurFY' or 'EPSNextFY'. See the Remarks for other typical Items and their Data strings.

Important!

GetFundamentalDetail has the ability to synchronize to only one same-type *Item* on

a given bar. For example, if two insiders traded on the same day (or within the same week/month for weekly/monthly bars), it is possible to access only the first of those transactions using **GetFundamentalDetail**. Of the fundamental items that contain detail strings, only 'estimated_earnings' is guaranteed to be unique by bar. If it is important in your analysis to access every item on each bar, then use the **FundamentalItemData** method and manually parse the *Data* string instead. Furthermore, only **FundamentalItemData** has the ability to access future (or past) earnings that do not fall within a chart's range.

Remarks

- **Wealth-Lab Pro only.** See the *Fundamental Data and Economic Indicator Definitions Guide* for valid *Item* parameters.
- **GetFundamentalDetail** always returns a string value. If necessary, convert the string to a number using **StrToInt** or **StrToFloat**.
- Only the following list of *Items* contain Data strings:
 - 'estimated_earnings' Data string (typical):
FY=2006;CurQTR=3;CurQTRMonth=9;CalendarYear=2006;EPSCurFY=2.57;EPSNextFY=2.874;
 - 'insider_transactions' Data string(typical):
FirmName=Market
Edge;NormalizedRating=SELL;ActionCode=DOWNGRADE;PrevNormalizedRating=NEUTRAL;AnalystName=Market Edge;
 - 'analyst_rating' Data string (typical):
transtype=B;insider=DION KURCZEK;title=Vice President;
- **GetFundamentalDetail** returns an empty string for valid *Items* that do not include details, e.g., 'assets', 'cash', 'dividend', etc.
- If *Item* is not found, the function raises an error, which can be detected using a try/except/end block (see "Error Handling" in the WealthScript Language Guide).
- To access fundamental data of secondary symbols, call **SetPrimarySeries** first.

Example

```
const EE = 'estimated_earnings';
var b: integer;
var eps, calyr: string;

// Get the most recent occurrence on the chart
b := BarCount - 1;
eps := GetFundamentalDetail( b, EE, 'EPSCurFY' );
calyr := GetFundamentalDetail( b, EE, 'CalendarYear' );
ShowMessage( 'The EPS of the current fiscal year is $' + eps + #13#10 +
'from calendar year ' + calyr );
```

9 Math Functions

9.1 Overview

Most scientific Math function that you would expect from a scripting language are available in WealthScript. If you cannot find the function that you're looking for, try browsing the Studies folder or Wealth-Lab [Code Library](#) found on the Wealth-Lab site.

Three of the math functions are specific to WealthScript and allow you to determine values or the location of a line drawn on the chart: [LineExtendX](#)^[66], [LineExtendY](#)^[66], and [TrendLineValue](#)^[73]. Note that these functions are applicable in a linear sense, and therefore should not be used for semi-logarithmic charting. See, for example, [LineExtendYLog](#) in the Code Library.

9.2 Abs

Abs(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the absolute value of the specified *Value*.

Example

```
{ Report on the change in price after 200 days }
var S: string;
var DIFF: float;
var BAR: integer;
Bar := BarCount - 1;
Diff := PriceClose( Bar ) - PriceClose( Bar - 200 );
Diff := Diff / PriceClose( Bar - 200 ) * 100;
s := 'After 200 days, prices ';
if Diff > 0 then
    s := s + 'advanced'
else
    s := s + 'declined';
Diff := Abs( Diff );
s := s + ' by ' + FormatFloat( '#0.0%', Diff );
DrawLabel( s, 0 );
```

9.3 ArcCos

ArcCos(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

ArcCos returns the inverse cosine of the specified number, *Value*. The number must be between -1 and 1. The return value is the angle, in degrees.

9.4 ArcSin

ArcSin(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

ArcSin returns the inverse sine of the specified number. The number must be between -1 and 1. The return value is the angle in degrees.

9.5 ArcSinh

ArcSinh(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

ArcSinh returns the inverse hyperbolic sine of the specified number, *Value*.

9.6 ArcTan

ArcTan(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Calculates the arctangent of a specified number, *Value*, in degrees.

9.7 ArcTanh

ArcTanh(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the inverse hyperbolic tangent of the specified number, *Value*. The number must be between -1 and 1.

9.8 Correlation

Correlation(Series1: integer; Series2: integer; StartBar: integer; EndBar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns Pearson's Correlation Coefficient between the two specified Price Series, *Series1* and *Series2*. Specify the quantity of data to analyze in the *StartBar* and *EndBar* parameters.

Example

```
{ How well correlated were CMO and RSI? }  
var corr: float;  
var RSISer, CMOSer: integer;  
RSISer := RSISeries( #Close, 20 );  
CMOSer := CMOSeries( #Close, 20 );
```

```
corr := Correlation( RSISer, CMOSer, 0, BarCount - 1 );
DrawLabel( 'Correlation: ' + FloatToStr( corr ), 0 );
```

9.9 Cos

Cos(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the cosine of the specified angle, *Value*. The angle should be specified in degrees.

9.10 Cosh

Cosh(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the hyperbolic cosine of the specified angle, *Value*.

9.11 Cotan

Cotan(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the cotangent of the specified angle. Specify the angle in degrees. Only use with angles that are non-zero.

9.12 Dec

Dec(Value: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Decrements the variable passed in the Value parameter by 1.

Note: Dec operates on regular integer variables only, not object field variables.

In the example below, the statement

```
Dec( cnt );
```

is equivalent to

```
cnt := cnt - 1;
```

Example

```
{ Each time the price crosses above 25, decrement a counter variable }
var Bar, cnt: integer;

cnt := 1000; // Initialize the variable
for Bar := 1 to BarCount - 1 do
```

```
if CrossOverValue( Bar, #Close, 25 ) then
    Dec( cnt );

ShowMessage( 'The counter is ' + IntToStr( cnt ) );
```

9.13 DegToRad

DegToRad(Degrees: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the value of the specified degree measurement, *Degrees*, in radians. The conversion from degrees to radians is given by the formula:

$$\text{radians} = \text{degrees} \left(\frac{\pi}{180} \right)$$

9.14 Exp

Exp(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the value of e raised to the specified *Value*, where e is the base of the natural logarithms and is approximately equal to 2.71828.

9.15 Frac

Frac(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the fractional part of the floating point number specified by *Value*.

9.16 Hypot

Hypot(x: float; y: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the length of the hypotenuse of a triangle. Specify the lengths of the sides adjacent to the right angle in *X* and *Y* parameters.

9.17 Inc

Inc(Value: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Increments the variable passed in the *Value* parameter by 1.

Note: `Inc` operates on regular integer variables only, not object field variables.

In the example below, the statement

```
Inc( cnt );
```

is equivalent to

```
cnt := cnt + 1;
```

Example

```
{ Each time the price crosses below 25, increment a counter variable }
var Bar, cnt: integer;

cnt := 0; // Initialize the variable
for Bar := 1 to BarCount - 1 do
  if CrossUnderValue( Bar, #Close, 25 ) then
    Inc( cnt );

ShowMessage( 'The counter is ' + IntToStr( cnt ) );
```

9.18 Int

Int(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the integer part of a floating point number **as a floating point number** after rounding towards zero. Assign the result of **Int** to a variable of type **float**.

Compare to: `Trunc`, `Round`

Example

```
{ f equals -5.0 following the conversion }
var x, f: float;
x := -5.678;
f := Int( x );
ShowMessage( FormatFloat( '0.0', f ) );
```

9.19 LinearRegLine

LinearRegLine(Series: integer; Start: integer; End: integer; Predict: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Allows you to perform ad-hoc linear regression analysis on the specified Price *Series*. Specify the *Start* and *End* bars for which to calculate the regression line. Then, specify the bar, *Predict*, for which you want to predict a value. This could be a bar that extends into the future.

Example

```
{ Draw a bullseye around the predicted closing price of the last bar of
the
  chart based on a linear regression that completed 1 - bars earlier }
var X: float;
var ENDBAR, BAR1, BAR2: integer;
```

```

EndBar := BarCount - 1;
Bar1 := EndBar - 30;
Bar2 := EndBar - 10;
x := LinearRegLine( #Close, Bar1, Bar2, EndBar );
DrawCircle( 8, 0, EndBar, x, #Red, #Thick );
DrawCircle( 4, 0, EndBar, x, #Black, #Thick );

```

9.20 LineExtendX

LineExtendX(x1: float; y1: float; x2: float; y2: float; y: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Extends the line specified by the $x1$, $y1$, $x2$, and $y2$ parameters, solving for x using the specified y parameter.

Remarks

- The equation used in the solution assumes a linear (not logarithmic) y -scale axis.

Example

```

{ Determine middle bar between last 2 peaks }
var PRICE1, PRICE2, PRICE3: float;
var BAR, BAR1, BAR2, BAR3: integer;
Bar := BarCount - 1;
Bar1 := PeakBar( Bar, #High, 13 );
Price1 := Peak( Bar, #High, 13 );
Bar2 := PeakBar( Bar1, #High, 13 );
Price2 := Peak( Bar1, #High, 13 );
Price3 := ( Price1 + Price2 ) / 2;
Bar3 := Trunc( LineExtendX( Bar1, Price1, Bar2, Price2, Price3 ) );

SetBarColor( Bar3, #Red );
DrawLine( Bar1, Price1, Bar2, Price2, 0, #Blue, #Thin );

```

9.21 LineExtendY

LineExtendY(x1: float; y1: float; x2: float; y2: float; x: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Extends the line specified by the $x1$, $y1$, $x2$, and $y2$ parameters and solves for y using the specified x parameter.

Remarks

- The equation used in the solution assumes a linear (not logarithmic) y -scale axis. See **LineExtendYLog** in the Wealth-Lab Code Library for the semi-log complementary function.
- See Also: **TrendLineValue**

Example

```

{ Extend recent resistance line to most current bar }
var PRICE1, PRICE2, Price3, Rev: float;
var BAR1, BAR, BAR2: integer;
Rev := 5;

```

```

Bar := BarCount - 1;
Bar1 := PeakBar( Bar, #High, Rev );
Price1 := Peak( Bar, #High, Rev );
Bar2 := PeakBar( Bar1, #High, Rev );
Price2 := Peak( Bar1, #High, Rev );
Price3 := LineExtendY( Bar1, Price1, Bar2, Price2, BarCount - 1 );

DrawLine( Bar1, Price1, Bar2, Price2, 0, #Blue, #Thick );
DrawLine( Bar2, Price2, BarCount - 1, Price3, 0, #Red, #Thin );

```

9.22 LN

LN(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the natural log of the specified *Value*.

Example

```

{ This function returns the log BaseN of a Number }
function Log( Number, BaseN: float ): float;
begin
    Result := LN( Number ) / LN( BaseN );
end;

var f: float;
f := Log( 81, 3 );
ShowMessage( 'The log base 3 of 81 is ' + FormatFloat( '0.0', f ) );

```

9.23 Log10

Log10(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the base 10 logarithm for the specified *Value*.

9.24 Log2

Log2(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the base 2 logarithm for the specified *Value*.

9.25 Max

Max(n1: float; n2: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the greater of the two specified values, *n1* and *n2*. The result is returned as a **float** type.

The ChartScript below is set up for optimization on two variables, which are used for the moving average periods. To ensure that both MA Price Series are valid for all permutations of the #OptVars, we use **Max** to determine which one is greater and use the result as the first bar in the main loop.

Example

```
{ Long-only moving avg crossover trading script set up for Optimization
}

{#OptVar1 8;6;14;2}
{#OptVar2 14;8;20;2}
var Bar, StartBar, p1, p2, hMA1, hMA2: integer;

p1 := #OptVar1;
p2 := #OptVar2;
hMA1 := SMASeries( #Close, p1 );
hMA2 := SMASeries( #Close, p2 );

{ Trunc converts the float type to an integer type }
StartBar := Trunc( Max( p1, p2 ) );
for Bar := StartBar to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    if CrossOver( Bar, hMA1, hMA2 ) then
      BuyAtMarket( Bar + 1, 'XOver' );
    end
  else
    if CrossUnder( Bar, hMA1, hMA2 ) then
      SellAtMarket( Bar + 1, LastPosition, 'XUnder' );
    end;
  PlotSeries( hMA1, 0, #Green, #Thin );
  PlotSeries( hMA2, 0, #Red, #Thin );
end;
```

9.26 Min

Min(n1: float; n2: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the lower of the two specified values, *n1* and *n2*. The result is returned as a **float** type.

In the example, the short-only trading system sets a stop based on the value of two Price Series. We use **Min** to determine the lesser value of the two series for the trailing stop.

Example

```

{ Each time the price crosses below 25, increment a counter variable }
var Bar, p1, p2, hMA1, hMA2: integer;
var Stp: float;

p1 := 10;
p2 := 20;
hMA1 := SMASeries( #Close, p1 );
hMA2 := SMASeries( #Close, p2 );
PlotStops;

for Bar := p2 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    if CrossUnder( Bar, hMA1, hMA2 ) then
    begin
      ShortAtMarket( Bar + 1, 'XUnder' );
      { Initialize a stop 3% higher than entry }
      Stp := PositionEntryPrice( LastPosition ) * 1.03;
      CoverAtTrailingStop( Bar + 1, Stp, LastPosition, 'Cvr' );
    end;
  end
  else
  begin
    if CrossOver( Bar, hMA1, hMA2 ) then
      CoverAtMarket( Bar + 1, LastPosition, 'XOver' );
    else
    begin
      Stp := Min( @hMA1[Bar] * 1.02, @hMA2[Bar] * 1.01 );
      CoverAtTrailingStop( Bar + 1, Stp, LastPosition, 'TStop' );
    end;
  end;
end;
PlotSeries( hMA1, 0, #Green, #Thin );
PlotSeries( hMA2, 0, #Red, #Thin );

```

9.27 Pi

Pi: float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the ratio of a circle's circumference to its diameter, approximately 3.141592854.

9.28 Power

Power(Base: float; Exponent: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Power raises the number specified by *Base* to the power specified in *Exponent*.

Note: For a fractional *Exponent*, *Base* must be greater than zero, otherwise a runtime error will result.

Remarks

To take the negative of the root $1/r$ of some positive number p , you can use:

```
var x, p: float;
p := 5;
x := ( -1 ) * Power( p, 1/3 )
```

Note that this is not the same as taking the negative root $1/r$ of p as in:

```
x = Power( -p, 1/r );
```

which results in a complex number and a run-time error.

9.29 RadToDeg

RadToDeg(Radians: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Converts the specified radian angle measurement, *Radians*, to degrees. The conversion from radians to degrees is given by the formula:

```
degrees = radians * ( 180 / pi )
```

9.30 RandG

RandG(Mean, StdDev: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Generates random numbers with a Gaussian distribution about the *Mean*. **RandG** is useful for simulating data with sampling errors and expected deviations from the *Mean*.

Remarks

- Call **SetRandSeed** before **RandG** to generate a repetitive random sequence.
- See also **Random**, **RandomInt**.

Example

```
{ Set 200 or 300 Fixed Bars and execute }
var n, m, BarCnt: integer;
var lst: TList = TList.Create;
var h: integer = CreateSeries;
var Pane: integer = CreatePane( 150, true, true );

BarCnt := BarCount - 1;
for n := 0 to BarCnt do
  lst.Add( RandG( 100, 5.0 ) );

{ Re-order and plot to show the Gaussian distribution }
lst.SortNumeric;
n := 0;
m := 0;
repeat
  @h[m] := lst.Item( n );
  Inc( n );
```

```

    @h[BarCnt - m] := lst.Item( n );
    Inc( m );
    Inc( n );
    until n >= BarCnt;
    PlotSeries( h, Pane, 0, #Histogram );

```

9.31 Random

Random: float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns a random number between zero and one.

Remarks

- Call **SetRandSeed** before **Random** to generate a repetitive random sequence.

Example

```

{ Get a random value between 100 and 200 }
var x: float;
x := Random * 100 + 100;

```

9.32 RandomInt

RandomInt(Limit: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns a random integer between zero and *Limit* - 1.

Remarks

- Call **SetRandSeed** before **RandomInt** to generate a repetitive random sequence.

Example

```

{ Get a random integer between 0 and 99 }
var ri: integer;
ri := RandomInt( 100 );
ShowMessage( IntToStr( ri ) );

```

9.33 Randomize

Randomize;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Initializes the random number generator with a random value. You can call this function at the start of a script to ensure that you get a different sequence of random numbers each time the script is executed.

9.34 RandSeed

RandSeed;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the random number generator's current "seed" value. You can use the **SetRandSeed** function to change the seed value, and start a repetitive sequence of random numbers.

9.35 Round

Round(Value: float): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Rounds the specified floating point number, *Value*, to the nearest whole number, which is returned as an **integer** type.

Remarks

- **Round** uses "Bankers Rounding", which means that if *Value* is exactly between two whole numbers, the result is always an even number.
- Compare to: **Int**, **Trunc**

Example

```
{ n equals 2 and p equals -3 at the end of the example }  
var x: float;  
var n, p: integer;  
x := 2.5;  
n := Round( x );  
x := -3.49;  
p := Round( x );  
ShowMessage( IntToStr(n) + #9 + IntToStr(p) );
```

9.36 SetRandSeed

SetRandSeed(Value: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Changes the seed *Value* of the random number generator. You can generate repetitive sequences of random numbers by resetting the seed to a set value.

9.37 Sin

Sin(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the sine of the specified angle, *Value*. The angle should be specified in degrees.

9.38 Sinh

Sinh(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the hyperbolic sine of the specified angle, *Value*.

9.39 Sqr

Sqr(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the square of the specified *Value*. The return value is *Value* * *Value*.

9.40 Sqrt

Sqrt(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the square root of the specified *Value*.

9.41 Tan

Tan(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the tangent of the specified angle, *Value*. The angle should be in degrees.

9.42 Tanh

Tanh(Value: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the hyperbolic tangent of the specified *Value*.

9.43 TrendLineValue

TrendLineValue(Bar: integer; TrendLine: string): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Allows you to obtain the value at *Bar* of a named *Trendline* that you drew with Wealth-Lab Developer 4.0's TrendLine tool. If the named *Trendline* could not be found for the current symbol and time frame, the function returns 0.

Remarks

- The equation used in the solution assumes a linear (not logarithmic) y-scale axis.
- See also: **LineExtendY**

Example

```
{ Have we crossed the resistance TrendLine? }
var RES: float;
var BAR: integer;
for Bar := 1 to BarCount - 1 do
begin
  res := TrendLineValue( Bar - 1, 'Resistance' );
  if PriceClose( Bar - 1 ) < res then
  begin
    res := TrendLineValue( Bar, 'Resistance' );
    if PriceClose( Bar ) >= res then
    begin
      SetBarColor( Bar, #Red );
      DrawCircle( 5, 0, Bar, res, #Red, #Thin );
    end;
  end;
end;
```

9.44 Trunc

Trunc(Value: float): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Truncates the decimal portion of the specified floating point number and returns the integer portion. **Trunc** returns an **integer** type.


Compare to: Int, Round

Example

```
{ n equals 1 at the end of the example }
var x: float;
var n: integer;
x := 1.234;
n := Trunc( x );
```

10 PerfScript Functions

10.1 Overview

PerfScripts, or **Performance Scripts**, are *Scriptable Performance Reports*. You can customize Wealth-Lab Performance Reports to display whatever performance metrics that you can imagine using the PerfScript feature. Performance Scripts must be saved to the special  **PerfScripts** folder, where a sample is included with your Wealth-Lab Developer 4.0 installation that duplicates the standard Wealth-Lab Performance Report.

When enabled in the ChartScript Window or \$simulator tools, Wealth-Lab will execute a PerfScript four times to process *All Trades (Long+Short)*, *Long Only*, *Short Only*, and *Buy & Hold* positions. Since Wealth-Lab automatically makes the appropriate group of positions available to the PerfScript during each of the four runs, it's not necessary to write special code to test position types.

Of the seven PerfScript functions, four are used to add data to a *performance record*, which is simply a single row of text in the Performance Report. Each row must have a unique *Label*. Depending on the type of data to be displayed, you'll reference this *Label* using either [PerfAddCurrency](#)^[77], [PerfAddNumber](#)^[77], [PerfAddPct](#)^[78], or [PerfAddString](#)^[78]. Consequently, the same performance record can display different types of data as required for *All Trades*, *Long Only*, etc.

For example, for any performance metric that involves a division, you should include logic to detect if the divisor is zero prior to the division operation. If it is, then you can use [PerfAddString](#)^[78] to show 'INF'. Otherwise, use one the other functions to display a number with the appropriate format.

10.2 AccountExposure

AccountExposure: float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the total market exposure in percent for the trading system. Wealth-Lab calculates exposure on a bar-by-bar basis and measures the area of the portfolio equity curve that was exposed to the market.

Remarks

- **AccountExposure** is available only for PerfScripts.

Example

```
PerfAddPct( 'Exposure', AccountExposure, 2, #Black, 0, 8 );
```

10.3 CashInterest

CashInterest: float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the total interest earned on uninvested cash during a \$imulation (\$imulator only).

Remarks

- **CashInterest** is available only for PerfScripts.
- Interest and loan options are found in the Options dialog, Trading Costs/Control options group.

Example

```
PerfAddCurrency( 'Cash Interest', CashInterest, #WinLoss, 0, 8 );
```

10.4 DividendsPaid

DividendsPaid: float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the total amount of dividends collected and paid during a \$imulation. The amount can be negative if dividends were paid while holding stock short. To enable dividend payment, mark the checkbox for "Apply Dividend Payments" in the Trading Costs/Control options dialog group.

Remarks

- **Wealth-Lab Pro only**. In "Developer" DividendsPaid always returns zero.
- **DividendsPaid** is available only for PerfScripts.

Example

```
PerfAddCurrency( 'Dividends', DividendsPaid, #WinLoss, 0, 8 );
```

10.5 MarginLoan

MarginLoan: float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the total cash paid out due to margin interest during a \$imulation (\$imulator only).

Remarks

- **MarginLoan** is available only for PerfScripts.
- Interest and loan options are found in the Options dialog, Trading Costs/Control options group.

Example

```
PerfAddCurrency( 'Margin Loan Interest', MarginLoan, #WinLoss, 0, 8 );
```

10.6 PerfAddCurrency

PerfAddCurrency(Label: string; Value: float; Color: integer; Style: integer; Size: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Adds a floating point value formatted using the *Decimal and Currency* configuration in the Options Dialog|System Settings.

<i>Label</i>	Text label identifying the performance record, displayed in the left-most column of the Performance report.
<i>Value</i>	Floating point expression of the performance metric to be displayed.
<i>Color</i>	Controls the <i>Value</i> 's text color using one of the standard color constants, e.g., #Black, #Red, #Green, etc. For in-the-black/in-the-red coloring (positive/negative values, respectively), use #WinLoss.
<i>Style</i>	Controls the style of the <i>Label</i> and can be either #Bold, #Italic, or 0 for normal type.
<i>Size</i>	Point size of font, 8 is standard.

Example

```
PerfAddCurrency( 'Net Profit', NetProfit, #WinLoss, 0, 8 );
```

10.7 PerfAddNumber

PerfAddNumber(Label: string; Value: float; Decimals: integer; Color: integer; Style: integer; Size: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Adds a floating point value, which is displayed with the specified number of *Decimals*. *Value* is rounded to correspond to the specified *Decimals* precision.

<i>Label</i>	Text label identifying the performance record, displayed in the left-most column of the Performance report.
<i>Value</i>	Floating point expression of the performance metric to be displayed.
<i>Decimals</i>	Number of digits to be displayed right of the decimal point. Rational numbers are zero-filled. Enter 0 for integer expressions.
<i>Color</i>	Controls the <i>Value</i> 's text color using one of the standard color constants, e.g., #Black, #Red, #Green, etc. For in-the-black/in-the-red coloring (positive/negative values, respectively), use #WinLoss.
<i>Style</i>	Controls the style of the <i>Label</i> and can be either #Bold, #Italic, or 0 for normal type.
<i>Size</i>	Point size of font, 8 is standard.

Example

```
PerfAddNumber( 'Number of Trades', PositionCount, 0, #Black, #Bold, 8 );
```

10.8 PerfAddPct

PerfAddPct(Label: string; Value: float; Decimals: integer; Color: integer; Style: integer; Size: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Adds a floating point value displayed with the % symbol. *Value* should already be formatted in percent terms, i.e., multiplied by 100.

<i>Label</i>	Text label identifying the performance record, displayed in the left-most column of the Performance report.
<i>Value</i>	Floating point expression of the performance metric to be displayed.
<i>Decimals</i>	Number of digits to be displayed right of the decimal point. Rational numbers are zero-filled. Enter 0 for integer expressions.
<i>Color</i>	Controls the <i>Value</i> 's text color using one of the standard color constants, e.g., #Black, #Red, #Green, etc. For in-the-black/in-the-red coloring (positive/negative values, respectively), use #WinLoss.
<i>Style</i>	Controls the style of the <i>Label</i> and can be either #Bold, #Italic, or 0 for normal type.
<i>Size</i>	Point size of font, 8 is standard.

Example

```
PerfAddPct( 'Net Profit %', ( NetProfit / StartingCapital ) * 100, 2, #WinLoss, #Bold, 8 );
```

10.9 PerfAddString

PerfAddString(Label: string; StringVal: string; Color: integer; Style: integer; Size: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Adds the string identified by the *StringVal* expression.

<i>Label</i>	Text label identifying the performance record, displayed in the left-most column of the Performance report.
<i>StringVal</i>	String expression or literal to be displayed.
<i>Color</i>	Controls <i>StringVal</i> 's text color using one of the standard color constants, e.g., #Black, #Red, #Green, etc.
<i>Style</i>	Controls the style of the <i>Label</i> and can be either #Bold, #Italic, or 0 for normal type.
<i>Size</i>	Point size of font, 8 is standard.

Example

```
PerfAddString( 'Annualized Gain $', 'N/A', #Black, 0, 8 );
```

10.10 PerfAddBreak

PerfAddBreak;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Adds a blank line after the last PerfAdd function call that creates a unique label.

Example

```
PerfAddBreak;
```

10.11 StartingCapital

StartingCapital: float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Starting Capital for Portfolio Simulations, or zero for Raw Profit Mode.

Remarks

- **StartingCapital** is available only for PerfScripts.

Example

```
PerfAddCurrency( 'Starting Capital', StartingCapital, #Black, #Bold, 8 );
```

10.12 TotalCommission

TotalCommission: float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the total commission generated during a simulation.

Remarks

- **TotalCommission** is available only for PerfScripts.
- The function is equally applicable to both the \$imulator and ChartScript Window tools in either Portfolio Simulation or Raw Profit mode.

Example

```
PerfAddCurrency( 'Total Commission', TotalCommission, #WinLoss, 0, 8 );
```

11 Position Management Functions

11.1 Overview

When you need to know about the performance or "properties" of an open or closed Position so to make a future trading decision, look to the Position Management category of functions. Each Position that you open will have a set of constant properties, such as the *Bar Number* on which the position was opened or the number of shares/contracts. All of these data are assigned to a *Position Number* that you later use to reference a particular Position. Position Numbers start at zero with the first Position opened by the ChartScript and increments by one for each newly opened (or split) Position.

While the Position is still open, you can also access up-to-the bar performance data, like Max Adverse/Favorable Excursions (MAE/MFE). Additionally, your Positions have extra storage for items such as a risk stop price ([SetPositionRiskStop](#)^[110]) for position sizing, a priority number to influence the decisions at the Portfolio Simulation level ([SetPositionPriority](#)^[108]), and an arbitrary data value ([SetPositionData](#)^[107]) that you can use for any purpose you choose!

11.2 ActivePositionCount

ActivePositionCount: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the number of active Positions currently being held by the trading system.

Example

```
{ Here we limit the system to 3 active Positions max }
var BAR, N: integer;

for Bar := 21 to BarCount - 1 do
begin
  n := 30;
  while n > 0 do
  begin
    if CrossUnderValue( Bar, RSISeries( #Close, 20 ), n ) then
      if ActivePositionCount < 3 then
        BuyAtMarket( Bar + 1, IntToStr( n ) );
    n := n - 5;
  end;
  if CrossOverValue( Bar, RSISeries( #Close, 20 ), 55 ) then
    for n := 0 to PositionCount - 1 do
      if PositionActive( n ) then
        SellAtMarket( Bar + 1, n, '' );
    end;
  var RSIPane: integer;
  RSIPane := CreatePane( 75, true, true );
  PlotSeries( RSISeries( #Close, 20 ), RSIPane, 205, #Thick );
  DrawLabel( 'RSI( Close, 20 )', RSIPane );
  AddScanColumn( 'RSI20', RSI( BarCount - 1, #Close, 20 ) );
```

11.3 ClearPositions

ClearPositions;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Clears out all Positions and Alerts. You'd only need to use this function if you're writing a complex script that is basing buy/sell decisions on an idealized profit curve. In this case, you could run one pass of the system and collect information in one or more in custom Price Series. You could then clear out the Positions and Alerts and execute the system again, this time using the metrics that you gathered in the first run.

Remarks

- In Portfolio Simulation Mode, **ClearPositions** resets **Equity** to the Starting Capital value in the Position Sizing Control, whereas Equity is reset to zero in Raw Profit Mode.

Example

```
{ Calculate the win/loss ratio of the system taking all trades.
  Then re-run the system, but only take trades when the prior
  win/loss ratio was above 50%. }
var WinLossPane, Winners, Trades, p, WinLoss, Bar, CMOPane: integer;

CMOPane := CreatePane( 80, true, true );
PlotSeries( CMOSeries( #Close, 20 ), CMOPane, 009, #Thick );
DrawLabel( 'CMO(Close,20)', CMOPane );

WinLoss := CreateSeries;

for Bar := 20 to BarCount - 1 do
begin
  Winners := 0;
  Trades := 0;
  for p := 0 to PositionCount - 1 do
  begin
    if not PositionActive( p ) then
    begin
      Inc( Trades );
      if PositionProfit( p ) > 0 then
        Inc( Winners );
    end;
  end;
  if Trades > 0 then
    SetSeriesValue( Bar, WinLoss, Winners * 100 / Trades );

  if CrossOverValue( Bar, CMOSeries( #Close, 20 ), -40 ) then
    BuyAtMarket( Bar + 1, 'CMO' );
  else if CrossUnderValue( Bar, CMOSeries( #Close, 20 ), 40 ) then
    SellAtMarket( Bar + 1, #All, 'CMO' );
end;

{ Plot the Win/Loss Ratio }
WinLossPane := CreatePane( 100, false, true );
SetPaneMinMax( WinLossPane, 0, 100 );
PlotSeries( WinLoss, WinLossPane, #Green, #ThickHist );
DrawLabel( 'Win/Loss Ratio', WinLossPane );

{ Clear the trades }
```

```

ClearPositions;

{ Execute the system again, but only take the trade if the
  win/loss ratio was above 50 }
for Bar := 20 to BarCount - 1 do
begin
  if CrossOverValue( Bar, CMOSeries( #Close, 20 ), -40 ) then
  begin
    if GetSeriesValue( Bar, WinLoss ) > 50 then
      BuyAtMarket( Bar + 1, 'CMO' );
    end
  else if CrossUnderValue( Bar, CMOSeries( #Close, 20 ), 40 ) then
    SellAtMarket( Bar + 1, #All, 'CMO' );
  end;
end;

```

11.4 GetPositionData

GetPositionData(Position: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the value of any user-specific data for this *Position* [Number]. You can store a single floating point value for each Position with the **SetPositionData** function in a ChartScript (only). If no user-specific data is stored, the function returns 0.

Note: In WLD3.0 and later, position data no longer influences \$imulator decisions during periods of insufficient capital. This task is now accomplished using **SetPositionPriority**.

Remarks

- Although you may assign data to a Position at any time after it has been created, the *ChartScript* must use **SetPositionData** on the *signal bar* if you plan to retrieve the data using **GetPositionData** in a SimuScript. Otherwise, using **SetPositionData** after the signal bar can result in a look-ahead (peeking) error during \$imulator processing.
- **GetPositionData** is available for use in a SimuScript referenced from the \$imulator only.
- In a SimuScript, pass the special constant **#Current** to access the *Position* that the \$imulator (or Portfolio Simulation) is currently sizing.

Example (SimuScript)

```

{ A ChartScript stores the current CMO level in the Position data.
  Use this value in a SimuScript to help establish Position size }
var x: float;
x := GetPositionData( #Current );
x := ( x + 100 ) / 2;
x := 100 - x;
SetPositionSizePct( x );

```

Example

```

{ This script uses Position Data to store whether the signal was
  generated from an RSI or an SMA Crossover. It uses the Position
  data to execute the corresponding exit. }
var P, BAR, CROSS: integer;
for Bar := 50 to BarCount - 1 do
begin

```

```

if CrossOverValue( Bar, RSISeries( #Close, 20 ), 30 ) then
begin
  BuyAtMarket( Bar + 1, 'RSI Cross 30' );
  SetPositionData( LastPosition, 1 );
end;
if CrossOver( Bar, SMASeries( #Close, 20 ), SMASeries( #Close, 50 ) )
then
begin
  BuyAtMarket( Bar + 1, 'SMA CrossOver' );
  SetPositionData( LastPosition, 2 );
end;
if CrossUnderValue( Bar, RSISeries( #Close, 20 ), 70 ) then
  for P := 0 to PositionCount - 1 do
    if PositionActive( P ) then
      if GetPositionData( P ) = 1 then
        SellAtMarket( Bar + 1, P, 'RSI Cross 70' );
  if CrossUnder( Bar, SMASeries( #Close, 20 ), SMASeries( #Close, 50 )
) then
    for P := 0 to PositionCount - 1 do
      if PositionActive( P ) then
        if GetPositionData( P ) = 2 then
          SellAtMarket( Bar + 1, P, 'SMA CrossOver' );
end;

```

11.5 GetPositionPriority

GetPositionPriority(Position: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Priority value that was assigned to the *Position* via a call to **SetPositionPriority**. A Position's priority determines whether or not it will be included by the portfolio simulator tool if there are more trades available than capital. Positions with a higher priority values take precedence. For example, a priority 5 Position will be included over a priority 1 Position if sufficient cash is not available for both Positions.

Remarks

- Priority values need not be simple integers, and, they can also be negative values.
- If Positions are not assigned priority by **SetPositionPriority**, Positions are chosen randomly when sufficient cash is not available for all trading signals during simulations. Note, however, that the random "seed" is always the same so that simulation results will be reproducible.

11.6 GetPositionRiskStop

GetPositionRiskStop(Position: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Risk Stop Level (the initial stop price) of the *Position*. The Risk Stop level is set by calling **SetRiskStopLevel** or **SetPositionRiskStop** from within a ChartScript (only).

Remarks

- Use this value in a SimuScript to determine a position size based on an initial stop loss level recorded in the ChartScript.

Example (SimuScript)

```

var XSTOP, XBASIS, XRISK: float;
xStop := GetPositionRiskStop( #Current );
xBasis := PositionBasisPrice( #Current );
xRisk := Abs( xBasis - xStop );
if xRisk > 10 then
  SetPositionSizeShares( 1 )
else if xRisk > 5 then
  SetPositionSizeShares( 2 )
else if xRisk > 2.5 then
  SetPositionSizeShares( 3 )
else if xRisk > 1 then
  SetPositionSizeShares( 4 )
else
  SetPositionSizeShares( 5 );

```

Example

```

{ Try and buy at the 20 bar low. If we get the trade, set a stop at the
  30 bar low. The $imulator can then use our stop level to create a
  position
  size that will risk whatever percent of capital we desire. }
var BAR, P: integer;
InstallProfitTarget( 10 );
for Bar := 0 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  if not LastPositionActive then
  begin
    SetRiskStopLevel( Lowest( Bar, #Low, 30 ) - 0.05 );
    BuyAtLimit( Bar + 1, Lowest( Bar, #Low, 20 ), '' );
  end
  else
  begin
    P := LastPosition;
    SellAtStop( Bar + 1, GetPositionRiskStop( P ), P, '' );
  end;
end;

```

11.7 LastActivePosition

LastActivePosition: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Position Number index of the *most current* active Position. If there are no active Positions, the function returns -1.

Remarks

- The list of positions contains all trades that have been created since the script first started executing. Each trade can be either open or closed. Closed means that the shares have been sold (or covered, for short positions). **LastActivePosition** returns the last open trade in the list.

- Avoid using *Last** functions in SimuScripts since they will not produce the desired result when used in the \$imulator.

Example

```
{ This simple average down system can be effective if you have
unlimited capital }
var BAR, P: integer;
for Bar := 20 to BarCount - 1 do
begin
  BuyAtLimit( Bar + 1, Lowest( Bar, #Close, 20 ), '' );
  if LastActivePosition >= 0 then
    for P := 0 to PositionCount - 1 do
      SellAtLimit( Bar + 1, Highest( Bar, #High, 13 ), P, '' );
    end;
end;
```

11.8 LastLongPositionActive

LastLongPositionActive: boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true if the last long Position is currently active.

Remarks

- The list of positions contains all trades that have been created since the script first started executing. Each trade can be either open or closed. If the last *long* position in the list is open, then **LastLongPositionActive** returns boolean *true*.
- Avoid using *Last** functions in SimuScripts since they will not produce the desired result when used in the \$imulator.

Caution! If your script uses a WatchList loop, it's possible that the **LastPosition** is from a symbol different to the one currently being processed. You can use the following function instead of **LastLongPositionActive** to ensure that you do not trade the last Position from the wrong symbol.

```
function LastLongPositionActiveSym( sym: string ): boolean;
begin
  Result := LastLongPositionActive;
  if Result then
    begin
      if PositionSymbol( LastPosition ) <> sym then
        Result := false;
      end;
    end;
end;
```

11.9 LastPosition

LastPosition: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Position Number of the most-recently created Position. This function is handy in Trading Systems that work with only one open Position at a time.

Remarks

- The list of positions contains all trades that have been created since the script first started executing. Each trade can be either open or closed. Closed means that the shares have been sold (or covered, for short positions). **LastPosition** returns the position number of the last trade in the list.
- **LastPosition** returns -1 if a position has not yet been created. Note that the number of the first position is 0.
- **LastPosition** is equivalent to (**PositionCount** - 1).
- Avoid using *Last** functions in SimuScripts since they will not produce the desired result when used in the \$imulator.

Example

```
var Bar: integer;
PlotSeries( SMASeries( #Close, 20 ), 0, #Red, #Thin );
PlotSeries( SMASeries( #Close, 10 ), 0, #Blue, #Thin );
for Bar := 40 to BarCount - 1 do
begin
  if CrossOver( Bar, SMASeries( #Close, 10 ), SMASeries( #Close, 20 ) )
  then
    BuyAtMarket( Bar + 1, '' )
  else if CrossOver( Bar, SMASeries( #Close, 20 ), SMASeries( #Close,
10 ) ) then
    SellAtMarket( Bar + 1, LastPosition, '' );
end;
```

11.10 LastPositionActive

LastPositionActive: boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true if the last Position is currently active. This is typically used in single Position trading systems to determine whether to execute the entry rules or the exit rules.

Remarks

- The list of positions contains all trades that have been created since the script first started executing. Each trade can be either open or closed. If the last position in the list (returned by **LastPosition**) is open, then **LastPositionActive** returns boolean *true*.
- Avoid using *Last** functions in SimuScripts since they will not produce the desired result when used in the \$imulator.

Caution! If your script uses a WatchList loop, it's possible that the **LastPosition** is from a symbol different to the one currently being processed. You can use the following function instead of **LastPositionActive** to ensure that you do not trade the last Position from the wrong symbol.

```
function LastPositionActiveSym( sym: string ): boolean;
begin
  Result := LastPositionActive;
  if Result then
  begin
    if PositionSymbol( LastPosition ) <> sym then
```

```

        Result := false;
    end;
end;

```

Example

```

var BAR: integer;
InstallStopLoss( 20 );
InstallProfitTarget( 7 );
for Bar := 40 to BarCount - 1 do
begin
    ApplyAutoStops( Bar );
    if not LastPositionActive then
        if CrossOverValue( Bar, RSISeries( #Average, 40 ), 25 ) then
            BuyAtLimit( Bar + 1, PriceHigh( Bar ), ' ' );
        end;
    end;
end;

```

11.11 LastShortPositionActive

LastShortPositionActive: boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true if the last short Position is currently active.

Remarks

- The list of positions contains all trades that have been created since the script first started executing. Each trade can be either open or closed. If the last *short* position in the list is open, then **LastShortPositionActive** returns boolean *true*.
- Avoid using *Last** functions in SimuScripts since they will not produce the desired result when used in the \$imulator.

Caution! If your script uses a WatchList loop, it's possible that the **LastPosition** is from a symbol different to the one currently being processed. You can use the following function instead of **LastShortPositionActive** to ensure that you do not trade the last Position from the wrong symbol.

```

function LastShortPositionActiveSym( sym: string ): boolean;
begin
    Result := LastShortPositionActive;
    if Result then
        begin
            if PositionSymbol( LastPosition ) <> sym then
                Result := false;
            end;
        end;
    end;
end;

```

11.12 MarketPosition

MarketPosition: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns 0 if the last position is closed, or a position has not yet been created. Returns 1 if the last position is active and long, and -1 if the last position is active and short. This function is useful for single-position systems that take long and short positions (see example).

Remarks

- Avoid using **MarketPosition** in SimuScripts since it will not produce the desired result when used in the \$imulator.

Example

```
{ Channel Breakout System for Futures }
var HH, LL, HL, LH, BAR: integer;
HH := HighestSeries( #High, 20 );
LL := LowestSeries( #Low, 20 );
HL := HighestSeries( #Low, 20 );
LH := LowestSeries( #High, 20 );
for Bar := 21 to BarCount - 1 do
begin
  case MarketPosition of
    0:
    begin
      if not BuyAtStop( Bar + 1, @HH[Bar], '' ) then
        ShortAtStop( Bar + 1, @LL[Bar], '' );
      end;
    1:
    begin
      if SellAtStop( Bar + 1, @LH[Bar], LastPosition, '' ) then
        ShortAtStop( Bar + 1, @LL[Bar], '' );
      end;
    -1:
    begin
      if CoverAtStop( Bar + 1, @HL[Bar], LastPosition, '' ) then
        BuyAtStop( Bar + 1, @HH[Bar], '' );
      end;
    end;
  end;
end;
```

11.13 PositionActive

PositionActive(Position: integer): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true if the specified Position is open, and false if it has been closed. Use this function in trading systems that manage multiple Positions.

Remarks

- For SimuScripts, **PositionActive** returns true if the specified *Position* is active at the time of the SimuScript call, otherwise false. You can use this function in Position Sizing scripts that are based on streaks, like the one that follows.

Example (SimuScript)

```
{ Use winning streaks of closed positions to establish a position size }
var STREAK, P: integer;
```

```

streak := 0;
for p := PositionCount - 1 downto 0 do
begin
  if not PositionActive( p ) then
  begin
    if PositionProfit( p ) <= 0 then
      Break
    else
      streak := streak + 1;
    end;
  end;
end;
if streak = 0 then
  streak := 1;
if streak > 10 then
  streak := 10;
SetPositionSizePct( streak * 10 );

```

Example

```

var BAR, P: integer;
for Bar := 40 to BarCount - 1 do
begin
  if CrossOverValue( Bar, RSISeries( #Average, 40 ), 35 ) then
    BuyAtLimit( Bar + 1, PriceHigh( Bar ), '' );
  if CrossUnderValue( Bar, RSISeries( #Average, 40 ), 70 ) then
    for P := 0 to PositionCount - 1 do
      if PositionActive( P ) then
        SellAtMarket( Bar + 1, P, '' );
    end;
end;

```

11.14 PositionBasisPrice

PositionBasisPrice(Position: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Basis Price of the specified *Position*. The Basis Price is the price used to establish the size of the *Position*.

Remarks

- For next-bar market orders, the Basis Price is the close of the bar on which the trade is signaled. The actual entry price is the open of the following bar.
- For AtClose orders, the Basis Price is equal to the closing price of the signal bar.
- For stop and limit orders, the Basis Price is equal to the Stop/Limit price. The trade may be filled at a different price if there is a gap against the Stop/Limit price.

Example (SimuScript)

```

var XSTOP, XBASIS, XRISK: float;
xStop := GetPositionRiskStop( #Current );
xBasis := PositionBasisPrice( #Current );
xRisk := Abs( xBasis - xStop );
if xRisk > 10 then
  SetPositionSizeShares( 1 )
else if xRisk > 5 then
  SetPositionSizeShares( 2 )
else if xRisk > 2.5 then
  SetPositionSizeShares( 3 )

```

```

else if xRisk > 1 then
  SetPositionSizeShares( 4 )
else
  SetPositionSizeShares( 5 );

```

Example

```

{ Display differences between Basis Price and Entry Price }
var Bar: integer;
PlotStops;
for Bar := 4 to BarCount - 1 do
begin
  if LastPositionActive then
    SellAtStop( Bar + 1, Lowest( Bar, #Low, 3 ), LastPosition, '' )
  else
    begin
      if BuyAtStop( Bar + 1, Highest( Bar, #High, 3 ), '' ) then
        Print( FloatToStr( PositionEntryPrice( LastPosition )
          - PositionBasisPrice( LastPosition ) ) );
    end;
end;

```

11.15 PositionBarsHeld

PositionBarsHeld(Position: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the number of bars the specified *Position* was active. This method is useful when calculating statistics at the *end of the script* and should not be used within the trading loop since it does *not* provide the number of bars currently held.

Remarks

- If the position was closed, **PositionBarsHeld** returns the same result obtained by subtracting **PositionEntryBar** from **PositionExitBar**.
- If the position is open after the trading loop ends, **PositionBarsHeld** returns the result obtained by subtracting **PositionEntryBar** from **BarCount - 1**.

11.16 PositionCount

PositionCount: integer;

ChartScripts *SimuScripts PerfScripts CMScripts

Description

Returns the total number of trading system Positions, both open and closed.

Remarks

- In SimuScripts, **PositionCount** returns the number of Positions at the time the SimuScript was called. This count does not include the Position that is currently being processed by the SimuScript.
- Since a Portfolio Simulation can reject trades for insufficient cash, **PositionCount** can differ from a Raw Profit PositionCount. Therefore in a SimuScript, **PositionCount** returns the number of positions that have been *accepted* by the current \$imulation run at the time the SimuScript is called.

Example (SimuScript)

```

{ Scale down Position size the more active Positions that we have }
var PS, P: integer;
ps := 40;
for p := 0 to PositionCount - 1 do
begin
  if PositionActive( p ) then
  begin
    ps := ps - 10;
    if ps = 10 then
      Break;
    end;
  end;
end;
SetPositionSizePct( ps );

```

The following ChartScript uses an advanced technique for quickly looping through all active Positions. Knowing that the most-recently created Positions have the greatest Position Numbers, we can "count backwards" and exit the PositionCount loop after determining that all active Positions have been processed. The result is a significant savings in processing time for ChartScripts that create many Positions since older, closed Positions are not processed needlessly.

Example

```

{ The script adds a long position whenever CumDown is 9 or greater.
  When CumUp = 9, all active positions are sold. }
var Bar, p, Processed, APCount: integer;
for Bar := 9 to BarCount - 1 do
begin
  if CumDown( Bar, #Close, 4 ) >= 9 then
    BuyAtMarket( Bar + 1, '' );
  if ( CumUp( Bar, #Close, 4 ) = 9 ) then
  begin
    APCount := ActivePositionCount;
    Processed := 0;
    for p := PositionCount - 1 downto 0 do
    begin
      if PositionActive( p ) then
      begin
        SellAtMarket( Bar + 1, p, '' );
        Inc( Processed );
      end;
    end;
    if Processed = APCount then
      break;
    end;
  end;
end;
end;

```

11.17 PositionEntryBar

PositionEntryBar(Position: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the bar number on which the *Position* was established.

Example

```

var Bar: integer;
for Bar := 20 to BarCount - 1 do

```

```

begin
  if not LastPositionActive then
    begin
      if StochK( Bar, 20 ) > 70 then
        BuyAtMarket( Bar + 1, 'Stoch' );
      end
    else
      begin
        { Sell after 10 days }
        if Bar - PositionEntryBar( LastPosition ) = 10 then
          SellAtMarket( Bar + 1, LastPosition, '10 day' );
        end;
      end;
    end;
  end;
end;

```

11.18 PositionEntryPrice

PositionEntryPrice(Position: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the entry price of the specified *Position*.

Remarks

- When backtesting, **PositionEntryPrice** returns the actual entry price for signals on Bar + 1 even though the order is executed on Bar. Do not use the entry price for a Position created on Bar for other trading rules on the same Bar.
- In general, SimuScripts should use **PositionBasisPrice**, not **PositionEntryPrice**, for the **#Current Position**. This is because at the time the SimuScript is called, only the basis price is known for next-bar orders. See the SimuScript example below.
- **PositionEntryPrice** always returns zero for entry signal Alerts.

Example (SimuScript)

```

{ Use a larger size if the Basis Price of the current Position
  is less than the average entry price of all Positions thus far. }
var p: integer;
var sumPr, avgPr: float;

if PositionCount = 0 then
  SetPositionSizeShares( 200 )
else
  begin
    for p := 0 to PositionCount - 1 do
      sumPr := sumPr + PositionEntryPrice( p );
    avgPr := sumPr / PositionCount;

    if PositionBasisPrice( #Current ) < avgPr then
      SetPositionSizeShares( 300 )
    else
      SetPositionSizeShares( 200 );
    end;
  end;
end;

```

Example

```

{ CMO Signals with Profit Target will open multiple Positions, but will
  wait until the price is lower than the previously established
  Position. }
var LOWESTPOSITION: float;

```

```

var NPANE, LASTBARBOUGHT, BAR, I: integer;

{ Plot 14 day CMO in new chart pane }
nPane := CreatePane( 60, TRUE, FALSE );
PlotSeries( CMOSeries( #Close, 20 ), nPane, 009, 0 );
DrawText( 'CMO 20', nPane, 4, 4, 006, 8 );
SetPaneMinMax( nPane, -60, 60 );
DrawHorzLine( 0, nPane, 666, 0 );
DrawHorzLine( 50, nPane, 666, 1 );
DrawHorzLine( -50, nPane, 666, 1 );

SetBarColors( #Black, #Black );
InstallProfitTarget( 10 );
LastBarBought := 0;
for Bar := 15 to BarCount - 1 do
begin
  if CMO( Bar, #Close, 20 ) <= -50 then
    SetBarColor( Bar, #Blue );
  else if CMO( Bar, #Close, 20 ) >= 50 then
    SetBarColor( Bar, #Red );
  ApplyAutoStops( Bar );
  if CMO( Bar, #Close, 20 ) > -50 then
    if CMO( Bar - 1, #Close, 20 ) <= -50 then
      begin
        LowestPosition := 9999.9;
        for i := 0 to PositionCount - 1 do
          if PositionActive( i ) then
            if PositionEntryPrice( i ) < LowestPosition then
              LowestPosition := PositionEntryPrice( i );
          if ( LowestPosition = 9999.9 ) or ( PriceClose( Bar ) <
LowestPosition ) then
            if Bar >= ( LastBarBought + 9 ) then
              begin
                BuyAtMarket( Bar + 1, '' );
                LastBarBought := Bar + 1;
              end;
            end;
          if CMO( Bar, #Close, 20 ) < 50 then
            if CMO( Bar - 1, #Close, 20 ) >= 50 then
              for i := 0 to PositionCount - 1 do
                if PositionActive( i ) then
                  SellAtMarket( Bar + 1, i, '' );
            end;
          end;
        end;
      end;
    end;
  end;
end;

```

11.19 PositionExitBar

PositionExitBar(Position: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the bar number on which the specified *Position* was closed. If *Position* is still active on the last bar of the chart, then **PositionExitBar** returns 0.

Example

```

{ Display the shortest and the longest holding time }
var BAR, LOWBAR, HIGHBAR, P, BARSHELD: integer;
for Bar := 20 to BarCount - 1 do
begin
  if not LastPositionActive then

```



```

begin
  if CrossUnderValue( Bar, RSISeries( #Close, 10 ), 20 ) then
    BuyAtMarket( Bar + 1, '' );
  end
  else
  begin
    if CrossOverValue( Bar, RSISeries( #Close, 10 ), 60 ) then
      SellAtMarket( Bar + 1, LastPosition, '' );
    end;
  end;
end;

LowBar := 0;
HighBar := 0;
for P := 0 to PositionCount - 1 do
begin
  BarsHeld := PositionExitBar( P ) - PositionEntryBar( P );
  if BarsHeld > HighBar then
    HighBar := BarsHeld;
  if ( BarsHeld < LowBar ) or ( LowBar = 0 ) then
    LowBar := BarsHeld;
  end;
  DrawLabel( 'Longest Holding Time: ' + IntToStr( HighBar ), 0 );
  DrawLabel( 'Shortest Holding Time: ' + IntToStr( LowBar ), 0 );
end;

```

11.20 PositionExitPrice

PositionExitPrice(Position: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the exit price of the specified *Position*.

Example

```

{ This procedure reports on entry and exit levels of all trades
  Note: #9 inserts a tab character and is equivalent to chr(9) }
procedure TradeReport;
begin
  var f, p: integer;
  var s: string;
  f := FileCreate( 'c:\trade report.txt' );
  for p := 0 to PositionCount - 1 do
  begin
    s := 'Entry:' + #9 + DateToStr( GetDate( PositionEntryBar( p ) ) )
      + #9 + FloatToStr( PositionEntryPrice( p ) ) + #9;
    s := s + 'Exit: ' + #9 + DateToStr( GetDate( PositionExitBar( p ) ) )
      + #9 + FloatToStr( PositionExitPrice( p ) );
    FileWrite( f, s );
  end;
end;

```

11.21 PositionExitSignalName

PositionExitSignalName(Position: integer): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the "Signal Name" of the exit signal for the specified *Position*. The Signal Name is always the last parameter (*SignalName*) of the **SellAt** or **CoverAt** function that closed the *Position*.

Tip: If your strategy has many different entries and exits, you use a Position's entry or exit signal name as a condition for future trading decisions, if desired.

Example

```
var BAR: integer;
InstallStopLoss( 20 );
InstallProfitTarget( 100 );
InstallTrailingStop( 10, 50 );
for Bar := 30 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  if CrossOverValue( Bar, RSISeries( #Close, 20 ), 28 ) then
    BuyAtMarket( Bar + 1, '' );
end;
if PositionCount > 0 then
  DrawLabel( 'Last Position was Closed by '
    + PositionExitSignalName( LastPosition ), 0 );
```

11.22 PositionLong

PositionLong(Position: integer): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true if the specified *Position* is long, and false if it is short.

Remarks

- To access the Position that the \$imulator (or Portfolio Simulation) is currently working with, use the special constant **#Current**.
- See also: **PositionShort**

Example

```
var BAR: integer;
for Bar := 5 to BarCount - 1 do
begin
  if LastPositionActive then
    if PositionLong( LastPosition ) then
      SellAtStop( Bar + 1, Lowest( Bar - 1, #Low, 4 ), LastPosition, '' );
    if not LastPositionActive then
      ShortAtStop( Bar + 1, Lowest( Bar - 1, #Low, 4 ), '' );
    else if LastPositionActive and not PositionLong( LastPosition ) then
      CoverAtStop( Bar + 1, Highest( Bar - 1, #High, 4 ), LastPosition, '' );
    if not LastPositionActive then
      BuyAtStop( Bar + 1, Highest( Bar - 1, #High, 4 ), '' );
end;
```

11.23 PositionMAE

PositionMAE(Position: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Max Adverse Excursion of a closed *Position*. Max Adverse Excursion is the worst loss that the Position experienced during its lifetime on an intraday basis. The result is reported in dollars, but you can use **PositionMAEPct** to return the MAE in percentage terms.

Remarks

- The MAE calculation uses the low of the *entry bar* for market or stop orders only. For limit and at close orders, MAE cannot be determined for the entry bar.
- Do not use **PositionMAE** for Positions that are still active. To obtain the MAE for an active Position, or the MAE for a particular bar, use the **PositionOpenMAE** function.
- Use **PositionMAE**, for example, at the end of a script in a PositionCount loop to collect trading statistics.
- Commissions are *not* considered in the MAE calculation.

Warning! Because it is a cash-based function, **PositionMAE** cannot be employed in a ChartScript's trading rules if destined for the \$imulator.

Example

```
var Bar, p: integer;
var ftmp: float;

for Bar := 20 to BarCount - 1 do
begin
  { Trading system rules }
end;

{ Find the average PositionMAE }
for p := 0 to PositionCount - 1 do
  ftmp := ftmp + PositionMAE( p );

ftmp := ftmp / PositionCount;
ShowMessage( 'Avg MAE = ' + FormatFloat( '0.00', ftmp ) );
```

11.24 PositionMAEPct

PositionMAEPct(Position: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Max Adverse Excursion of a closed *Position* in percent. Max Adverse Excursion is the worst loss that the Position experienced during its lifetime on an intraday basis. Use **PositionMAE** to return the MAE in dollar terms.

Remarks

- The MAE calculation uses the low of the *entry bar* for market or stop orders only. For limit and at close orders, MAE cannot be determined for the entry bar.
- Do not use **PositionMAEPct** for Positions that are still active. To obtain the percentage MAE for an active Position, or the percentage MAE for a particular bar, use the **PositionOpenMAEPct** function.
- Use **PositionMAEPct**, for example, at the end of a script in a PositionCount loop to collect trading statistics.
- Commissions are *not* considered in the MAEPct calculation.

Example

```
var Bar, p: integer;
var ftmp: float;

for Bar := 20 to BarCount - 1 do
begin
  { Trading system rules }
end;

{ Find the average PositionMAEPct }
for p := 0 to PositionCount - 1 do
  ftmp := ftmp + PositionMAEPct( p );

ftmp := ftmp / PositionCount;
ShowMessage('Avg MAE (%) = ' + FormatFloat( '0.00%', ftmp ) );
```

11.25 PositionMFE

PositionMFE(Position: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Max Favorable Excursion of a closed *Position*. Max Favorable Excursion is the largest gain that the Position experienced during its lifetime on an intraday basis. The result is reported in dollars, but you can use **PositionMFEpct** to return the MFE in percentage terms.

Remarks

- The MFE calculation uses the high of the *entry bar* for market or stop orders only. For limit and at close orders, MFE cannot be determined for the entry bar.
- Do not use **PositionMFE** for Positions that are still active. To obtain the MFE for an active Position, or the MFE for a particular bar, use the **PositionOpenMFE** function.
- Use **PositionMFE**, for example, at the end of a script in a PositionCount loop to collect trading statistics.
- Commissions are *not* considered in the MFE calculation.

Warning! Because it is a cash-based function, **PositionMFE** cannot be employed in a ChartScript's trading rules if destined for the \$imulator.

Example

```
var Bar, p: integer;
var ftmp: float;
```

```

for Bar := 20 to BarCount - 1 do
begin
{ Trading system rules }
end;

{ Find the average PositionMFE }
for p := 0 to PositionCount - 1 do
ftmp := ftmp + PositionMFE( p );

ftmp := ftmp / PositionCount;
ShowMessage( 'Avg MFE = ' + FormatFloat( '0.00', ftmp ) );

```

11.26 PositionMFEPct

PositionMFEPct(Position: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Max Favorable Excursion of a closed *Position* in percent. Max Favorable Excursion is the largest gain that the Position experienced during its lifetime on an intraday basis. Use **PositionMFE** to return the MFE in dollar terms.

This function returns zero for Positions that are still active. To obtain the percentage MFE for an active Position, or the percentage MFE for a particular bar, use the **PositionOpenMFEPct** function.

Remarks

- The MFE calculation uses the high of the *entry bar* for market or stop orders only. For limit and at close orders, MFE cannot be determined for the entry bar.
- Do not use **PositionMFEPct** for Positions that are still active. To obtain the percentage MFE for an active Position, or the percentage MFE for a particular bar, use the **PositionOpenMFEPct** function.
- Use **PositionMFEPct**, for example, at the end of a script in a PositionCount loop to collect trading statistics.
- Commissions are *not* considered in the MFEPct calculation.

Example

```

var Bar, p: integer;
var ftmp: float;

for Bar := 20 to BarCount - 1 do
begin
{ Trading system rules }
end;

{ Find the average PositionMFEPct }
for p := 0 to PositionCount - 1 do
ftmp := ftmp + PositionMFEPct( p );

ftmp := ftmp / PositionCount;
ShowMessage( 'Avg MFE (%) = ' + FormatFloat( '0.00%', ftmp ) );

```

11.27 PositionOpenMAE

PositionOpenMAE(Position: integer; Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Max Adverse Excursion of an open *Position*, or the MAE for a Position as of the specified *Bar*. Max Adverse Excursion is the worst loss that the Position experienced during its lifetime on an intraday basis. The result is reported in dollars (negative), use **PositionOpenMAEPct** to return the MAE of an open Position in percentage terms.

Remarks

- The MAE calculation uses the low of the *entry bar* for market or stop orders only. For limit and at close orders, MAE cannot be determined for the entry bar.
- Commissions are *not* considered in the MAE calculation.

Warning! Because it is a cash-based function, **PositionOpenMAE** cannot be employed in a ChartScript's trading rules if destined for the \$imulator.

Example

```
var Bar, p: integer;
var ePrice: float;

PlotStops;
for Bar := 50 to BarCount - 1 do
begin
  if LastPositionActive then
  begin { Exit rules }
    p := LastPosition;
    ePrice := PositionEntryPrice( p );

    { Forget about profit and initiate a break-even stop when MAE exceeds
    -$750 }
    if PositionOpenMAE( p, Bar ) < -750 then
      SellAtLimit( Bar + 1, ePrice, p, 'beStop' )
    else
      SellAtLimit( Bar + 1, ePrice * 1.25, p, 'Pft Tgt' );
    end
  else { Entry rule }
  if CrossOver( Bar, #Close, SMAseries( #Close, 50 ) ) then
    BuyAtMarket( Bar + 1, '' );
  end;
end;
```

11.28 PositionOpenMAEPct

PositionOpenMAEPct(Position: integer; Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Max Adverse Excursion of an open *Position*, or the MAE for a Position as of a specified *Bar*, in percent. Max Adverse Excursion is the worst loss that the Position experienced during its lifetime on an intraday basis. Use **PositionOpenMAE** to return the MAE of an open Position in dollar terms.

Remarks

- The MAE calculation uses the low of the *entry bar* for market or stop orders only. For limit and at close orders, MAE cannot be determined for the entry bar.
- Commissions are *not* considered in the MAEPct calculation.

Example

```

var Bar, p: integer;
var ePrice: float;

PlotStops;
for Bar := 50 to BarCount - 1 do
begin
  if LastPositionActive then
  begin { Exit rules }
    p := LastPosition;
    ePrice := PositionEntryPrice( p );

    { Forget about the 25% profit and initiate a break-even stop when
    MAEPct drops -15% }
    if PositionOpenMAEPct( p, Bar ) <= -15 then
      SellAtLimit( Bar + 1, ePrice, p, 'beStop' )
    else
      SellAtLimit( Bar + 1, ePrice * 1.25, p, 'Pft Tgt' );
    end
  else { Entry rule }
    if CrossOver( Bar, #Close, SMAseries( #Close, 50 ) ) then
      BuyAtMarket( Bar + 1, '' );
  end;
end;

```

11.29 PositionOpenMFE

PositionOpenMFE(Position: integer; Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Max Favorable Excursion of an open *Position*, or the MFE for a Position as of a specified *Bar*. Max Favorable Excursion is the largest gain that the Position experienced during its lifetime on an intraday basis. The result is reported in dollars, use **PositionOpenMFEPct** to return the MFE of an open Position in percentage terms.

Remarks

- The MFE calculation uses the high of the *entry bar* for market or stop orders only. For limit and at close orders, MFE cannot be determined for the entry bar.
- Commissions are *not* considered in the MFE calculation.

Warning! Because it is a cash-based function, **PositionOpenMFE** cannot be employed in a ChartScript's trading rules if destined for the \$imulator.

11.30 PositionOpenMFEPct

PositionOpenMFEPct(Position: integer; Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Max Favorable Excursion of an open *Position*, or the MFE for a Position as of a specified *Bar*, in percent. Max Favorable Excursion is the largest gain that the Position experienced during its lifetime on an intraday basis. Use **PositionOpenMFE** to return the MFE of an open Position in dollar terms.

Remarks

- The MFE calculation uses the high of the *entry bar* for market or stop orders only. For limit and at close orders, MFE cannot be determined for the entry bar.
- Commissions are *not* considered in the MFEPct calculation.

11.31 PositionOpenProfit

PositionOpenProfit(Bar: integer; Position: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the current profit level of the *Position* (in dollars) as of the specified *Bar*. If the Position is closed, and the Bar number is on or before the Bar on which that the Position was closed, this function returns the same value as **PositionProfit**.

Remarks

- The value reported for **PositionOpenProfit** reflects trading costs as of the specified *Bar*, i.e., one-sided commissions and slippage are deducted from profit (or loss) while the position is open.

Warning! Because it is a cash-based function, **PositionOpenProfit** cannot be employed in a ChartScript's trading rules if destined for the \$imulator.

Example

```
{ Record a position's open profit as a Price Series.
  This system buys on a crossover of a 30-period weighted
  moving average and sells after 20 bars. }
var Bar, ProfitPane, hMA, hPftSer: integer;
var fPft: float;

hMA := WMASeries( #Close, 30 );
hPftSer := CreateSeries;

InstallTimeBasedExit( 20 );
for Bar := 30 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  if not LastPositionActive then
  begin
    if CrossOverValue( Bar, #Close, @hMA[Bar-1] ) then
      BuyAtMarket( Bar + 1, 'Xover' );
    end
  else
    @hPftSer[Bar] := PositionOpenProfit( Bar, LastPosition );
  end;

ProfitPane := CreatePane( 75, true, true );
PlotSeriesLabel( hPftSer, ProfitPane, 009, #Histogram, 'Open Profit' );
```



```
PlotSeries( hMA, 0, 909, #Thin );
```

11.32 PositionOpenProfitPct

PositionOpenProfitPct(Bar: integer; Position: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the current percentage profit level of the *Position* as of the specified *Bar*. If the *Position* is closed, and the *Bar* number is on or before the *Bar* on which that the *Position* was closed, this function returns the same value as **PositionProfitPct**.

Remarks

- The value reported for **PositionOpenProfitPct** reflects trading costs as of the specified *Bar*, i.e, one-sided commissions and slippage are deducted from profit (or loss) while the position is open.

Warning! For non-zero commissions, **PositionProfitPct** is not compatible with the \$imulator.

Example

```
{ Record a position's open profit percentage as a Price Series.
  This system buys on a crossover of a 30-period weighted
  moving average and sells after 20 bars. }
var Bar, ProfitPane, hMA, hPftSer: integer;
var fPft: float;

hMA := WMASeries( #Close, 30 );
hPftSer := CreateSeries;

InstallTimeBasedExit( 20 );
for Bar := 30 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  if not LastPositionActive then
  begin
    if CrossOverValue( Bar, #Close, @hMA[Bar-1] ) then
      BuyAtMarket( Bar + 1, 'Xover' );
    end
  else
    @hPftSer[Bar] := PositionOpenProfitPct( Bar, LastPosition );
  end;

ProfitPane := CreatePane( 75, true, true );
PlotSeriesLabel( hPftSer, ProfitPane, 009, #Histogram, 'Open Profit
Pct' );
PlotSeries( hMA, 0, 909, #Thin );
```

11.33 PositionOrderType

PositionOrderType(Position: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns an integer indicating the type of order used for the specified *Position*.

0 = Market
 1 = Stop
 2 = Limit
 3 = Close

Remarks

- See **CMOrderType** for CommissionScripts.

Example

```
{ SimuScript:
  Vary the number of shares purchased based on the PositionOrderType }
case PositionOrderType( #Current ) of
  0, 1: // AtMarket, AtStop
    SetPositionSizeShares( 500 );
  2: // AtLimit
    SetPositionSizeShares( 300 );
  else
    SetPositionSizeShares( 100 );
end;
```

11.34 PositionProfit

PositionProfit(Position: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the net profit in dollars of the specified closed *Position*. Use **PositionProfitPct** to return the profit in percentage terms.

Remarks

- The value reported for **PositionProfit** is reduced by the trading costs, i.e, commissions and slippage.
- Use **PositionOpenProfit** to return the current profit of an open Position at a specific bar. The value reported by **PositionProfit** for an open Position is **invalid**.

Warning! Because it is a cash-based function, **PositionProfit** cannot be employed in a ChartScript's trading rules if destined for the \$imulator.

Example (SimuScript)

```
{ Use 20% sizing, but scale back to 10% if the last closed Position was
a loss }
var p: integer;
var x: float = 20;
for p := PositionCount - 1 downto 0 do
  if not PositionActive( p ) then
    begin
      if PositionProfit( p ) < 0 then
        x := 10;
      break;
    end;

  SetPositionSizePct( x );
```

Example

```

{ This function returns the average profit of the last "Num" Positions
}
function AvgProfit( Bar, Num: integer ): float;
begin
  var p: integer;
  var sump: float;
  sump := 0;
  for p := PositionCount - 1 downto PositionCount - Num do
    sump := sump + PositionProfit( p );
  Result := sump / Num;
end;

{ Make some arbitrary trades and call the function }
var Bar: integer;
var pft: float;
InstallProfitTarget( 5 );
InstallStopLoss( 2.5 );
for Bar := 20 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  if Bar Mod 50 = 0 then
    BuyAtMarket( Bar + 1, '' );

end;
pft := AvgProfit( BarCount - 1, PositionCount );
ShowMessage( 'The avg profit of the last '
  + IntToStr( PositionCount ) + ' trades was '
  + FormatFloat( '#,###.00', pft ) );

```

11.35 PositionProfitPct

PositionProfitPct(Position: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the percentage profit of the specified closed *Position*. Use **PositionProfit** to return the profit in dollar terms.

Remarks

- The value reported for **PositionProfitPct** reflects trading costs, i.e, commissions and slippage are deducted from profit (or loss).
- Use **PositionOpenProfitPct** to return the current percentage profit of an open Position at a specific bar. The value reported by **PositionProfitPct** for an open Position is **invalid**.

Warning! For non-zero commissions, **PositionProfitPct** is not compatible with the \$imulator.

Example

```

{ SimuScript: Use average Position Profit to determine Position size }
var XSUM, X: float;
var N, P: integer;
x := 10;
n := 0;
xsum := 0;
for p := PositionCount - 1 downto 0 do
begin

```

```

if not PositionActive( p ) then
begin
  n := n + 1;
  xsum := xsum + PositionProfitPct( p );
  if n = 10 then
    Break;
  end;
end;
if xsum > 0 then
begin
  x := xsum / n;
  if x < 10 then
    x := 10;
  end;
  SetPositionSizePct( x );

```

11.36 PositionShares

PositionShares(Position: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the number of shares (or contracts) in the specified Position.

Example

```

{ Write number of shares of open Positions to debug window }
procedure WriteOpenTrades;
begin
  var i: integer;
  for i := 0 to PositionCount - 1 do
    if PositionActive( i ) then
      Print( IntToStr( PositionShares( i ) ) + ' Shares ' + GetSymbol
    );
  end;

```

11.37 PositionShort

PositionShort(Position: integer): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true if the specified *Position* is a short Position.

Remarks

- To access the Position that the \$imulator (or Portfolio Simulation) is currently working with, use the special constant **#Current**.
- See also: **PositionLong**

The example shows how you can use this function in a **SimuScript**.

Example

```
{ Risk half as many shares for short positions }
if PositionShort( #Current ) then
  SetPositionSizeShares( 100 )
else
  SetPositionSizeShares( 200 );
```

11.38 PositionSignalName

PositionSignalName(Position: integer): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the "Signal Name" of the entry signal for the specified *Position*. The Signal Name is always the last parameter (*SignalName*) of the **BuyAt** or **ShortAt** function that opened the *Position*.

Remarks

- As shown in the example, you can use **PositionSignalName** to execute different exit strategies based on the entry strategy that was used to open the Position.
- In a SimuScript, you can use the **PositionSignalName** to size a *Position* based on the strategy that was used to open the Position, for example.
- By formatting a string with multiple delimited fields, you can pass many types of data via *SignalName*, retrieve them with **PositionSignalName**, and parse the result with **GetToken**.

Example

```
{ The following script combines an RSI and a CMO strategy into a single system }
var Bar, p: integer;
for Bar := 20 to BarCount - 1 do
begin
  { Exit logic }
  for p := 0 to PositionCount - 1 do
    if PositionActive( p ) then
      begin
        if PositionSignalName( p ) = 'RSI' then
          begin
            if CrossOverValue( Bar, RSISeries( #Close, 20 ), 60 ) then
              SellAtMarket( Bar + 1, p, 'RSI' );
          end
        else if CrossOverValue( Bar, CMOSeries( #Close, 20 ), 50 ) then
              SellAtMarket( Bar + 1, p, 'CMO' );
        end;
      end;
  { Entry logic }
  if CrossOverValue( Bar, RSISeries( #Close, 20 ), 30 ) then
    BuyAtMarket( Bar + 1, 'RSI' );
  if CrossOverValue( Bar, CMOSeries( #Close, 20 ), -50 ) then
    BuyAtMarket( Bar + 1, 'CMO' );
end;
```

11.39 PositionSymbol

PositionSymbol(Position): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the symbol of the selected *Position*.

Example

```
{ SimuScript:
  Ensure only 1 trade per symbol }
var p: integer;
var s: string;
s := PositionSymbol( #Current );
SetPositionSizePct( 10 );
for p := 0 to PositionCount - 1 do
  if PositionSymbol( p ) = s then
    if PositionActive( p ) then
      SetPositionSizeFixed( 0 );
```

11.40 SetPositionData

SetPositionData(Position: integer; Value: float);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Lets you store user-specific data (a single floating point number), *Value*, for the specified *Position*. You can obtain the Position data value using **GetPositionData**. Use these functions if you need to track any additional information on Positions that you need to act on in your script.

For instance, your trading system might take an initial trend-following position, and then subsequent scalping positions to try and nab quick profits. You can use **SetPositionData** to record which type of Positions were taken so that you could apply the appropriate exit logic.

Remarks

- Although you may assign data to a Position at any time after it has been created, the *ChartScript* must use **SetPositionData** on the *signal bar* if you plan to retrieve the data using **GetPositionData** in a *SimuScript*. Otherwise, using **SetPositionData** after the signal bar can result in a look-ahead (peeking) error during \$imulator processing.
- Wealth-Lab processes sizing before actually creating Positions in the ChartScript Window and Scans, therefore *Value* (the data) from **SetPositionData** is not available to SimuScripts in these tools. Instead, you can pass data using *SignalName* and retrieve it with **PositionSignalName** in the SimuScript. *SignalName* is the last string parameter in the **BuyAt** and **ShortAt** signals, and it may contain all kinds of information, which you can parse using **GetToken**, if required.

Example

```
{ This seasonal system stores the month of entry for use later in a
  SimuScript }
var BAR: integer;
for Bar := 40 to BarCount - 1 do
```

```

begin
  if LastPositionActive then
    SellAtLimit( Bar + 1, Highest( Bar, #High, 20 ), LastPosition, '' )
  else
    if BuyAtLimit( Bar + 1, Lowest( Bar, #Low, 20 ), '' ) then
      SetPositionData( LastPosition, GetMonth( Bar ) );
    end;
  end;
end;

```

11.41 SetPositionPriority

SetPositionPriority(Position: integer; Priority: float);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Sets the *Priority* value of a *Position*. A Position's priority determines whether or not it will be included by the Portfolio \$imulator tool if there are more trades available than capital for a specific bar. Positions with a higher value for *Priority* take precedence, i.e., they are processed first. For example, a Position with *Priority* 10.5 will be processed before any Position having *Priority* less than 10.5. In most cases, it's not required to set priority, and in this case the priority is random. However, specifying priority is useful, for example, to force the \$imulator to choose a "most/least condition", where *condition* may be oversold, lowest-priced, etc.

SetPositionPriority is designed for ChartScripts that use **Buy/ShortAtMarket (or AtClose) entries**. For example, assume that your trading system generates 10 orders to place on the next bar, but you have cash enough for 4 orders only. *Prior to placing orders*, you can decide which of the orders to place based on some indicator or price.

AtLimit/AtStop Entry Orders

Generally speaking, you should not **SetPositionPriority** for ChartScripts that use AtLimit/AtStop entries. Doing so may create a peeking effect since it's not possible to know which limit (or stop) orders will execute first when orders are placed for multiple instruments.

Exceptions:

1. If the ChartScript employs a "multi-dip buyer" strategy, use **SetPositionPriority** to assign higher priority to AtLimit orders with higher limit prices, for example. If you don't, the possibility exists for the the \$imulator to execute orders with lower limit prices first (and vice-versa for ShortAtLimit).
2. You can intentionally peek to determine if an AtLimit/AtStop order occurred at the opening price, and in this case you could assign a priority of 1 to these Positions. This is a valid backtesting method, demonstrated by the following simple ChartScript.

Warning! You *must* employ **SetPositionPriority** in ChartScripts that use multiple order-entry types, such as AtMarket and AtLimit orders. Since the \$imulator does not distinguish between the types, set a higher priority for AtMarket entries so that they are processed before AtLimit/AtStop orders on the same bar.

```

var Bar: integer;
var limitprice, priority: float;

InstallTimeBasedExit( 5 );
for Bar := 10 to BarCount - 1 do

```

```

begin
  ApplyAutoStops( Bar );
  if not LastPositionActive then
    begin
      limitprice := Lowest( Bar, #Low, 10 );
      priority := 0;

      { peek to check if limit order will be executed "at market" on the
        open }
      if Bar < BarCount - 1 then
        if PriceOpen( Bar + 1 ) <= limitprice then
          priority := 1;

          if BuyAtLimit( Bar + 1, limitprice, '' ) then
            SetPositionPriority( LastPosition, priority );
          end;
        end;
      end;
    end;
  end;
end;

```

Remarks

- Use **SetPositionPriority** on the *signal bar*. Once you have assigned priority, it should not be changed on subsequent bars.
- Use the boolean return value of entry signals (especially for **AtLimit/AtStop** orders) as a condition to calling **SetPositionPriority**, e.g.,


```

{ After a limit buy, give priority to the position having the lowest
  RSI }
if BuyAtLimit( Bar + 1, LimitPrice, '' ) then
  SetPositionPriority( LastPosition, -1 * RSI(Bar, #Close, 14 ) );

```
- Use **GetPositionPriority** to obtain the priority of a Position.
- Position priority is used only in the \$imulator and Portfolio \$imulation Mode in the Optimizer Control.

Example

```

{ Buy when CMO is oversold and assign the highest priority
  to the most oversold by multiplying the CMO value by -1 }
var BAR, hRSI, hCMO, CP: integer;
hCMO := CMOSeries( #Close, 20 );
CP := CreatePane( 75, true, true );
PlotSeriesLabel( hCMO, CP, #Blue, #Thin, 'CMO(20)' );

for Bar := 20 to BarCount - 1 do
begin
  if not LastPositionActive then
    begin
      if @hCMO[ Bar ] < -55 then
        begin
          if BuyAtMarket( Bar + 1, 'CMO' ) then
            SetPositionPriority( LastPosition, -1 * @hCMO[ Bar ] );
          end;
        end;
      end;
    end;
  else if @hCMO[ Bar ] > 45 then
    SellAtMarket( Bar + 1, LastPosition, 'CMO' );
  end;
end;

```


11.42 SetPositionRiskStop

SetPositionRiskStop(Position: integer; StopLevel: float);

ChartScripts SimuScripts PerfScripts CMScripts

Note: **SetRiskStopLevel** supersedes the **SetPositionRiskStop** function. See **SetRiskStopLevel** for information.

Description

Specifies the initial stop level, *StopLevel*, for the *Position*. This stop level is used in the \$imulator and other tools in Portfolio Simulation mode when you select the **Maximum Risk Pct** Position Sizing option. When this option is selected, you tell the Portfolio Simulation the percentage of capital you are willing to risk on the trade. The Portfolio Simulation then uses the Position's RiskStop value to determine how many shares to assign to the Position.

Important: When you use **Maximum Risk Pct** sizing, you're responsible for actually exiting the Position at the stop level in your ChartScript code.

Example

```
{ Set our risk stop at 50 cents below the signal day's low }
var BAR: integer;
InstallProfitTarget( 50 );
for Bar := 40 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  if LastPositionActive then
    SellAtStop( Bar + 1, GetPositionRiskStop( LastPosition ),
LastPosition, '' )
  else
    if BuyAtLimit( Bar + 1, Lowest( Bar, #Low, 20 ), '' ) then
      SetPositionRiskStop( LastPosition, PriceLow( Bar ) - 0.50 );
end;
```

11.43 SetRiskStopLevel

SetRiskStopLevel(StopLevel: float);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Specifies the initial *StopLevel* (price) for the next Position to be created. This stop level is used in the \$imulator and other tools in Portfolio Simulation mode when you select the **Maximum Risk Pct** Position Sizing option. When this option is selected, you tell the Portfolio Simulation the percentage of capital you are willing to risk on the trade. The Portfolio Simulation then uses the Position's RiskStop value to determine how many shares to assign to the Position.

Remarks

- When you use **Maximum Risk Pct** sizing, you're responsible for actually exiting the Position at the stop level in your ChartScript code.
- For automated trading, you can use **SetRiskStopLevel** to automatically activate a stop loss order *on the same bar* on which a position is entered. See "Automated Trading Options" in the User Guide for more information.

Note: **SetRiskStopLevel** supersedes the original **SetPositionRiskStop** function. The problem with **SetPositionRiskStop** is that the Position must already be created. This is fine in the \$imulator, which works on a list of trades, but is not possible at the ChartScript level, where the initial stop level must be known *before* the Position is created. Using **Maximum Risk Pct** position sizing at the ChartScript level is new to Wealth-Lab Developer 3.0, so this new function was introduced to support it.

Example

```
{ Set our risk stop at 50 cents below the signal day's low }
var BAR: integer;
InstallProfitTarget( 50 );
for Bar := 40 to BarCount - 1 do
begin
    ApplyAutoStops( Bar );
    if LastPositionActive then
        SellAtStop( Bar + 1, GetPositionRiskStop( LastPosition ),
LastPosition, '' )
    else
    begin
        SetRiskStopLevel( PriceLow( Bar ) - 0.5 );
        BuyAtLimit( Bar + 1, Lowest( Bar, #Low, 20 ), '' );
    end;
end;
```

12 Price Series Functions

12.1 Overview

There's no use in avoiding it. If you're going to deal with a technical application for market analysis, you're going to be working with *Price Series*. Generally speaking a Price Series refers to an array of values that has the same number of elements as bars loaded in a chart. The Price Series category of functions allow you to create, analyze, synchronize, change time frames, and otherwise manipulate an entire series of data with a minimal amount of effort. Most of these functions automatically create a new (result) Price Series and return a *handle* that you use to refer to the new series.

A subset of these functions (CrossOver/Under and TurnUp/Down) provide relative information between two prices, or possibly between a Price Series and a fixed value. In this sense, they act in an "indicative" fashion, though they are not what we consider a true indicator.

12.2 AbsSeries

```
AbsSeries( Series: integer ): integer;
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns a new Price Series that contains the absolute values of the Price Specified in the *Series* parameter.

Example

```
var DIFF, ABSDIFF, P: integer;
Diff := SubtractSeries( #Open, #Close );
AbsDiff := AbsSeries( Diff );
p := CreatePane( 100, true, true );
PlotSeries( Diff, p, #Black, #Thin );
PlotSeries( AbsDiff, p, #Red, #Thin );
```

12.3 AddCalendarDays

```
AddCalendarDays( Interpolate: boolean );
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Adds all missing calendar days to the chart data, including weekends, holidays, and any other non-trading day. Newly added bars are considered "synthetic", and these bar numbers return true when **SyntheticBar** is called.

The value of the inserted bars depends on the *Interpolate* parameter. If *Interpolate* is false, the new bars assume the OHLC values of the previous bar. If *Interpolate* is true, the OHLC values of the new bars are calculated using linear interpolation between the previous bar and the next actual bar. Note that interpolating values will result in the bars being created based on future information (next bar's value) so be careful if using these bars in trading system development.

Remarks

- **AddCalendarDays** is compatible with the Daily scale only.
- **AddCalendarDays** is not compatible with Real-Time Scans or Real-Time ChartScript windows.

Example

```
var Bar: integer;
DrawLabel( 'BarCount Before: ' + IntToStr( BarCount ), 0 );
if IsDaily then
    AddCalendarDays( false );
DrawLabel( 'BarCount After: ' + IntToStr( BarCount ), 0 );
for Bar := 0 to BarCount - 1 do
    if SyntheticBar( Bar ) then
        SetBarColor( Bar, #Red );
```

12.4 AddFutureBars

AddFutureBars(Bars: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Adds the number of bars specified in the *Bars* parameter to the end of the chart. The added bars are considered "synthetic", and calling **SyntheticBar** with these bar numbers will return true. The OHLC values of the added bars are taken from the last actual bar in the chart. You can change the values of these (or any) bars by using the **ChangeBar** method.

Note: **AddFutureBars** is not currently compatible with Real-time Scans or ChartScript windows.

If **AddCalendarDays** had previously been called, the future bars will include non-trading days. If **AddCalendarDays** had not been called, the future bars will not include weekend days (Saturday, Sunday).

Remarks

- Functions that make changes to the Primary Data Series, such as **ChangeBar** and **AddFutureBars**, should not be used in scripts opened for Optimization.
- **AddFutureBars** is not available for use on the web site, only in Wealth-Lab Developer 4.0.

Warning! Adding futures bars will affect when Alerts are issued for an established trading system. Alerts are generated when a trading signal occurs after the last bar of the chart, i.e., on bar number *BarCount*. This means that if a system developer is using future bars, Alerts will also be projected into the future by the number of bars added.

Example

```
var Bar: integer;
DrawLabel( 'BarCount Before: ' + IntToStr( BarCount ), 0 );
if IsDaily then
    AddCalendarDays( false );
AddFutureBars( 20 );
DrawLabel( 'BarCount After: ' + IntToStr( BarCount ), 0 );
for Bar := 0 to BarCount - 1 do
```

```

if SyntheticBar( Bar ) then
  SetBarColor( Bar, #Red );

```

12.5 AddSeries

AddSeries(Series1: integer; Series2: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Adds the two specified Price Series, *Series1* and *Series2*, and returns the handle to a new Price Series.

Example

```

{ Plot the mean price }
var ADDED, MEAN: integer;
Added := AddSeries( #High, #Low );
Mean := DivideSeriesValue( Added, 2 );
PlotSeries( Mean, 0, #Blue, #Thick );

```

12.6 AddSeriesValue

AddSeriesValue(Series: integer; Value: float): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Adds a constant *Value* to each element of the Price Series, *Series*, and returns the handle to a new Price Series.

Example

```

{ This script creates a normalized MACD by adding the lowest value,
  bringing all of values in the series above zero. }
var X: float;
var MSER, MPANE: integer;
MSer := MACDSeries( #Close );
x := Abs( Lowest( BarCount - 1, MSer, BarCount ) );
MSer := AddSeriesValue( MSer, x );
MPane := CreatePane( 100, true, true );
PlotSeries( MSer, MPane, #Red, #Histogram );

```

12.7 AnalyzeSeries

AnalyzeSeries(PriceSeries: integer; Description: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Outputs the value of the Price Series defined by its handle, *PriceSeries*, to the Trade Log at the time the trade was signaled. The *PriceSeries* is also available on the "Analysis" tab of the ChartScript and \$imulator windows for more detailed analysis.

Note: AnalyzeSeries is not compatible in the with ChartScripts that execute trades on symbols other than the primary symbol, i.e., following a call to **SetPrimarySeries**. Additionally and generally speaking, scripts with WatchList loops should not be used in the \$imulator.

You can use **AnalyzeSeries** to determine how an indicator might be applied to an existing trading system to filter losing trades without having to actually incorporate the indicator into the system. In the example below, trading signals are based solely on the **StochRSISeries** indicator. At the end of the script, a **CMOSeries** indicator is created for analysis.

Example

```
{ Buy when StochRSI turns up from 0 and sell when turns down from 100 }
var Bar, StochRSIPane, hStRSI: integer;
var cmoSer, cmoPane: integer;

hStRSI := StochRSISeries( #Close, 14 );
InstallBreakEvenStop( 5 );
PlotStops;
for Bar := 30 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  if LastPositionActive then
  begin
    if @hStRSI[ Bar - 1 ] >= 99.9 Then
    if TurnDown( Bar, hStRSI ) then
      SellAtMarket( Bar + 1, LastPosition, '' );
    end
  else
    if @hStRSI[ Bar - 1 ] <= 0.1 Then
    if TurnUp( Bar, hStRSI ) then
      BuyAtMarket( Bar + 1, '' );
    end;
  end;
  StochRSIPane := CreatePane( 75, true, true );
  PlotSeriesLabel( StochRSISeries( #Close, 14 ), StochRSIPane, 411, 2,
    'StochRSI(14)' );

  cmoPane := CreatePane( 75, true, true );
  cmoSer := CMOSeries( #Close, 14 );
  PlotSeriesLabel( cmoSer, cmoPane, #Green, #Thin, 'CMO 14' );
  AnalyzeSeries( cmoSer, 'CMO 14' );
end;
```

12.8 ChangeBar

ChangeBar(Bar: integer; Date: integer; Time: integer; Open: float; High: float; Low: float; Close: float; Volume: integer; OpenInterest: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Provides the capability to change all values for the specified *Bar*'s properties: *Date*, *Time*, *Open*, *High*, *Low*, *Close*, *Volume*, and *OpenInterest*. This is most useful in assigning values to forecasted bars that have been added using the **AddFutureBars** method.

The *Date* parameter should be a standard WealthScript date integer, e.g., 1/15/2003 = 20030115. Likewise, for intraday charts, the *Time* parameter should be a standard WealthScript time integer, e.g. 14:00 = 1400. Otherwise, specify zero for *Time* when using non-intraday charts.

Remarks

Functions that make changes to the Primary Data Series, such as **ChangeBar** and **AddFutureBars**, should not be used in scripts opened for Optimization.

Example

```

{ Add a copy of the last 10 bars to the end of the chart }
var Bar, Cbars, b: integer;
Cbars := 10;
AddFutureBars( Cbars );
for Bar := BarCount - Cbars to BarCount - 1 do
begin
  b := Bar - Cbars;
  SetBarColor( Bar, #Blue );
  ChangeBar( Bar, GetDate( b ), GetTime( b ), PriceOpen( b ),
    PriceHigh( b ), PriceLow( b ), PriceClose( b ),
    Trunc( Volume( b ), Trunc( OpenInterest( b ) ) );
end;

```

12.9 ClearExternalSeries

```
ClearExternalSeries( Symbol: string );
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Clears any external (secondary) price series from memory. External series are obtained from calls to either **GetExternalSeries** or **SetPrimarySeries**. The *Symbol* parameter is optional. If it is specified, only the external series of the specified symbol will be cleared. If you pass a blank string instead, all external series accessed so far will be cleared.

This function was introduced as a way to optimize scripts that process large lists of symbols. These scripts can quickly bog down because all of the external series accessed remain in memory until the script is completed. By calling **ClearExternalSeries**, these scripts can free resources that are no longer being used, resulting in better script performance.

Remarks

- Do not use **ClearExternalSeries** for a *Symbol* on which you've created trades.

Example

```
ClearExternalSeries( '' );
```

12.10 ClearIndicators

```
ClearIndicators;
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Clears any indicators that have been created from memory. This function is useful for scripts that cycle through all of the symbols in a WatchList and process each symbol in turn, creating indicators for each symbol. Scripts like this can easily run out of resources because the indicator series are being re-created, and the old series remain in memory.

Example

```

var RSISer, w, Bar: integer;
for w := 0 to WatchListCount - 1 do
begin

```

```

PrintStatus( WatchListSymbol( w ) );
ClearIndicators;
for Bar := 0 to BarCount - 1 do
begin
  SetPrimarySeries( WatchListSymbol( w ) );
  RSISer := RSISeries( #Close, 20 );
  { ... do more processing here ... }
end;
end;

```

12.11 CreateNamedSeries

CreateNamedSeries(SeriesName: string): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Creates a new blank price series and returns the new series as a PriceSeries variable, like **CreateSeries**. You can use the **FindNamedSeries** function to locate a named series that you previously created. This pair of functions is used internally by the New Indicator Wizard.

Example

```

var n: integer;
n := CreateNamedSeries( 'MySeries' );

```

12.12 CreateSeries

CreateSeries: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Creates a new blank Price Series and returns the handle of the new Series. The new Price Series contains the number of values equal to the ChartScript's BarCount. The values in the Series are initially set to zero. Use **SetSeriesValue** to load values in the new series and **GetSeriesValue** to read the values (or use the shorthand @ syntax).

Remarks

- Generally speaking, use **CreateSeries** outside of loops and then fill the series with values in a loop as shown in the example.

Example

```

{ The following script creates a new Price Series, and stores the
  difference between the 20 day high and the 20 day low. }
var X: float;
var N, BAR, MYPANE: integer;
n := CreateSeries;
for Bar := 20 to BarCount - 1 do
begin
  x := Highest( Bar, #High, 20 ) - Lowest( Bar, #Low, 20 );
  SetSeriesValue( Bar, n, x );
end;
MyPane := CreatePane( 100, true, true );
PlotSeries( n, MyPane, #Green, #ThickHist );

```


12.13 CreateSeriesLength

CreateSeriesLength(Length: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Creates a new *Length*-sized Price Series and returns the handle of the new Series. The new Price Series contains the number of values equal to the *Length* parameter, indexed from zero. The values in the Series are initially set to zero. Use **SetSeriesValue** to set values and **GetSeriesValue** to read values (or use the shorthand @ syntax).

CreateSeriesLength provides the ability to create a *Length*-sized array at run-time; in other words, a pseudo-dynamic array.

Example

```
var x, hMySer, SerLength: integer;
var f: float;

SerLength := StrToInt( Input( 'Type in an integer number' ) );
hMySer := CreateSeriesLength( SerLength );

for x := 0 to SerLength - 1 do
begin
    @hMySer[x] := x * 1.5;
    Print( IntToStr( x ) + #9 + FloatToStr( @hMySer[x] ) );
end;
```

12.14 CrossOver

CrossOver(Bar: integer; Series1: integer; Series2: integer): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true when *Series1* crosses above the *Series2* at *Bar*.

Specifically, **CrossOver** triggers (true) if both of the following conditions are true:

```
@Series1[Bar] > @Series2[Bar], AND,
@Series1[Bar - 1] <= @Series2[Bar - 1]
```

Example

```
{ A simple Weighted Moving Average Crossover System }
var BAR: integer;
InstallProfitTarget( 6 );
InstallStopLoss( 4 );
for Bar := 60 to BarCount - 1 do
begin
    ApplyAutoStops( Bar );
    if CrossOver( Bar, WMASeries( #Close, 30 ), WMASeries( #Close, 60 ) )
then
    BuyAtMarket( Bar + 1, '' );
end;
```

12.15 CrossOverValue

CrossOverValue(Bar: integer; Series: integer; Value: float): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true when the Price *Series* crosses above the specified *Value* at *Bar*.

Specifically, **CrossOver** triggers (true) if both of the following conditions are true:

```
@Series[Bar] > Value, AND,
@Series[Bar - 1] <= Value
```

Example

```
{ A Basic "Extreme RSI" type System }
var BAR: integer;
InstallProfitTarget( 5 );
InstallStopLoss( 20 );
for Bar := 30 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  if CrossOverValue( Bar, RSISeries( #Close, 32 ), 24 ) then
    BuyAtMarket( Bar + 1, 'Extreme RSI' );
end;
```

12.16 CrossUnder

CrossUnder(Bar: integer; Series1: integer; Series2: integer): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true when *Series1* crosses below *Series2* at *Bar*.

Specifically, **CrossUnder** triggers (true) if both of the following conditions are true:

```
@Series1[Bar] < @Series2[Bar], AND,
@Series1[Bar - 1] >= @Series2[Bar - 1]
```

Example

```
{ This system opens a new position whenever Stochastic crosses above
  its signal line from below 20. It closes all positions when
  Stochastic crosses below the signal line from above 80. }
var STOCHPANE, SIGNAL, BAR, P: integer;
StochPane := CreatePane( 150, true, true );
Signal := EMASeries( StochDSeries( 20, 3 ), 9 );
PlotSeries( StochDSeries( 20, 3 ), StochPane, 009, #Thin );
PlotSeries( Signal, StochPane, #Gray, #Thin );

for Bar := 30 to BarCount - 1 do
begin
  if CrossUnder( Bar, StochDSeries( 20, 3 ), Signal ) then
    if StochD( Bar - 1, 20, 3 ) > 80 then
      for P := 0 to PositionCount - 1 do
        if PositionActive( P ) then
          SellAtMarket( Bar + 1, P, 'Stoch' );
    if CrossOver( Bar, StochDSeries( 20, 3 ), Signal ) then
      if StochD( Bar - 1, 20, 3 ) < 20 then
        BuyAtMarket( Bar + 1, 'Stoch' );
```

```
end;
```

12.17 CrossUnderValue

CrossUnderValue(Bar: integer; Series: integer; Value: float): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true when Price Series crosses below the specified Value at Bar.

Specifically, **CrossUnder** triggers (true) if both of the following conditions are true:

```
@Series[Bar] < Value, AND,
@Series[Bar - 1] >= Value
```

Example

```
{ This system buys as soon as DSS crosses below 30 }
var DSSPANE, BAR: integer;
DSSPane := CreatePane( 100, true, true );
PlotSeries( DSSSeries( 10, 20, 5 ), DSSPane, 905, #Thick );
InstallStopLoss( 5 );
InstallProfitTarget( 15 );
for Bar := 20 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  if CrossUnderValue( Bar, DSSSeries( 10, 20, 5 ), 30 ) then
    BuyAtMarket( Bar + 1, 'DSS' );
end;
```

12.18 DivideSeries

DivideSeries(Series1: integer; Series2: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Divides each element of Series1 by its corresponding element in Series2, and returns the handle to the resulting Price Series.

Example

```
{ Display Relative Strength of DJIA to Nasdaq }
var NAZ, DJ, RS, RSPANE: integer;
Naz := GetExternalSeries( '^IXIC', #Close );
DJ := GetExternalSeries( '^DJI', #Close );
RS := DivideSeries( DJ, Naz );
RSPane := CreatePane( 100, true, true );
PlotSeries( RS, RSPane, #Black, #Thick );
DrawLabel( 'Strength Relative of DJIA to Nasdaq', RSPane );
```

12.19 DivideSeriesValue

DivideSeriesValue(Series: integer; Value: float): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Divides each element of the specified *Price Series* by a constant *Value* and returns the handle to a new *Price Series*. You should ensure that *Value* is non-zero.

Example

```
{ Divide RSI by 100 to get it in the range of 0 to 1 }
var RSIRANGE, RSIPANE: integer;
RSIRange := DivideSeriesValue( RSIseries( #Close, 30 ), 100 );
RSIPane := CreatePane( 100, true, true );
PlotSeries( RSIRange, RSIPane, #Teal, #Thick );
```

12.20 DivideValueSeries

DivideValueSeries(Value: float; Series: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Divides a constant *Value* by the value of each element in the specified *Price Series* and returns the handle to a new *Price Series*.

Note: Be careful not to confuse this with **DivideSeriesValue**.

12.21 EnableSynch

EnableSynch(Enable: boolean);

ChartScripts SimuScripts PerfScripts CMScripts

Description

If *Enable* is false, **EnableSynch** will turn off the default synchronization that occurs for external series in a **ChartScript**. An external series is any series referenced through a call to **GetExternalSeries** or **SetPrimarySeries**.

The default synchronization options are controlled via the Options Dialog. When "Enable Automatic Date/Time Synchronization of External Series" is checked, the synchronization process truncates any secondary series that has more bars than the primary series (the symbol that was clicked to execute the script). For detailed information on "How Secondary Data Series are Synchronized in Wealth-Lab", see the article by the same name on the Wealth-Lab Articles page.

You can use **EnableSynch** to turn off the default synchronization behavior set in the Options Dialog. If you do this, you should use the **SynchAll** or **SynchSeries** statements in your script to perform synchronization manually. Manual synchronization will perfectly align any secondary series to the primary, by removing dates that don't exist in the primary, and inserting dates in the secondary that exist in the primary but not the secondary. By calling **SynchAll** or **SynchSeries** after any indicators are created on the secondary series, you can be sure that the series is aligned correctly AND that the indicator values are calculated correctly, taking into account all bars in the pre-aligned secondary series.

Example

```
{ Print the BarCount of each Symbol }
var w: integer;
EnableSynch( false );
```

```
for w := 0 to WatchListCount - 1 do
begin
  SetPrimarySeries( WatchListSymbol( w ) );
  Print( WatchListSymbol( w ) + #9 + IntToStr( BarCount ) );
end;
```

12.22 FindNamedSeries

FindNamedSeries(SeriesName: string): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the integer handle of the Price Series with the specified name, *SeriesName*. If no PriceSeries with the given name was found, the function returns -1. This will be the PriceSeries that was created in one of the following manners:

- by a corresponding call to **CreateNamedSeries**.
- by adding a Custom Field during the creation of an ASCII DataSource. Pass the "Field Name" that you entered when creating the DataSource.

This function is used internally in the code created by the New Indicator Wizard, and, unless you have additional names fields for ASCII DataSources, you should rarely need to call this function yourself.

Remarks

- ASCII Custom Fields work only in the chart data's native time frame. For example, if you have a daily ASCII DataSource, the Custom Field data is not available if you switch to Weekly scale from the toolbar.
- The string passed at *SeriesName* is case sensitive. For example, if the custom field was defined as 'PE_Ratio', passing 'PE_RATIO' will cause the function to fail. You can review the actual field name by clicking the Properties button for the DataSource in the DataSource Manager.

Example

```
var MySeries: integer;
MySeries := FindNamedSeries( 'SeriesName' );
```

12.23 FirstActualBar

FirstActualBar: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

This function is useful in scripts that loop through and execute trades on all of the symbols in a WatchList. In these cases, Wealth-Lab's synchronization feature (see **Tools|Options|Synchronization**) will transform secondary data series so that they synchronize with the Primary series, the one clicked to run the script. If a secondary data series has a shorter history than the Primary series, data bars are appended to the beginning of the secondary series so that its BarCount equals that of the Primary series. **FirstActualBar** will return the bar number that represents the first "real" bar of the secondary series. You can use this value to make sure that you don't enter trades on the symbol before its actual history began.

Example

```

var FIRST: integer;
SetPrimarySeries( 'ABGX' );
First := FirstActualBar;
RestorePrimarySeries;
DrawLabel( 'ABGX started trading on bar ' + IntToStr( First ), 0 );

```

12.24 GetDescription

GetDescription(Series: integer): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the internal description of the specified Price *Series*. This function is used in the New Indicator Wizard to create a properly named Price Series for a new indicator being created.

Example

```

var e: integer;
e := EMASeries( #Close, 60 );
PlotSeries( e, 0, 009, #Thin );
DrawLabel( GetDescription( e ), 0 );

```

12.25 GetExternalSeries

GetExternalSeries(Symbol: string; Series: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the requested Price *Series* for the specified *Symbol*. *Series* must be one of the following Standard Price Series constants (only):

#Open, #High, #Low, #Close, #Volume, and #OpenInterest

Remarks

- Wealth-Lab will first try to find the *Symbol* in the WatchList of the primary symbol, which is the one that was clicked. If it doesn't exist in the primary symbol's WatchList, it will try to find it in other WatchLists/DataSources, top to bottom, alphabetically. Use **AllowSymbolSearch** to limit symbol searches to specific WatchLists.
- You cannot call **GetExternalSeries** after you have changed the Primary Series with **SetPrimarySeries**, or after you have re-scaled the Primary Series with a Time Frame function.
- **GetExternalSeries** generates an error at run time if the *Series* cannot be found. If there's reason to suspect a *Symbol* will not be found, you can "catch" the error with a try/except block as shown here:

```

var h: integer;
try
  h := GetExternalSeries( 'MSFT', #Close );
except
  Print( 'No data or could not find series' );
end;

```

- If you require access to an external symbol's Custom Field(s), do not use **GetExternalSeries**. Instead, use **SetPrimarySeries**, followed by **FindNamedSeries**, and finally **RestorePrimarySeries**.
- External Series functionality is not available for Tick or Second-based charts.

Example

```
{ Display price of this series relative to CSCO }
var CSCO, REL, RELPANE: integer;
CSCO := GetExternalSeries( 'CSCO', #Close );
Rel := DivideSeries( #Close, CSCO );
RelPane := CreatePane( 100, false, true );
PlotSeries( Rel, RelPane, #Teal, #Thick );
```

12.26 GetSeriesValue

GetSeriesValue(Bar: integer; Series: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns a value at a specified *Bar* from the specified Price *Series*. Use this function to return values from a Price Series that you create with **CreateSeries**, or the result of any WealthScript function that returns a Price Series integer handle.

Remarks

Instead of using **GetSeriesValue**, you can use the @-symbol shorthand notation. In the example below, the statement

```
Per := Round( GetSeriesValue( Bar, AdaptivePer ) );
```

is equivalent to

```
Per := Round( @AdaptivePer[ Bar ] );
```

Note: The @ syntax is not compatible with Price Series whose handles are stored in a declared array; e.g., `@h[i][Bar]`, where `h[i]` is an integer array of Price Series handles, is not valid syntax. See the WealthScript Language Guide for more information.

Example

```
{ Create an adaptive moving average by multiplying a base period by R-Squared }
var ADAPTIVEPER, ADAPTIVEMA, BAR, PER: integer;
AdaptivePer := CreateSeries;
AdaptiveMA := CreateSeries;
AdaptivePer := MultiplySeriesValue( RSquaredSeries( #Close, 30 ), 60 );
for Bar := 60 to BarCount - 1 do
begin
  Per := Round( GetSeriesValue( Bar, AdaptivePer ) );
  if Per < 5 then
    Per := 5;
  SetSeriesValue( Bar, AdaptiveMA, SMA( Bar, #Close, Per ) );
end;
PlotSeries( AdaptiveMA, 0, #Purple, #Thick );
```

12.27 MultiplySeries

MultiplySeries(Series1: integer; Series2: integer): integer;

ChartScripts SimuScripts PerfScripts CMScritps

Description

Multiplies each element of two Price Series, *Series1* and *Series2*, returning the handle to a new Price Series.

Example

```
var ROCPANE, ROCSMA, ROCSMA1, ROCSMA2, SMAMULT: integer;

{ Plot 30 Day SMA }
PlotSeries( SMAseries( #Close, 30 ), 0, 040, #Thick );
DrawText( '30 Day SMA', 0, 4, 46, 040, 8 );

{ Create a De-Trended 30 Day SMA by accounting for SMA slope }
RocPane := CreatePane( 75, true, true );
RocSMA := ROCseries( SMAseries( #Close, 30 ), 1 );
PlotSeries( RocSMA, RocPane, 020, #ThickHist );
DrawText( '1 Day ROC of 30 Day SMA', RocPane, 4, 4, 020, 8 );

RocSMA1 := MultiplySeriesValue( RocSMA, 0.1 );
RocSMA2 := AddSeriesValue( RocSMA1, 1.0 );
SMAMult := MultiplySeries( SMAseries( #Close, 30 ), RocSMA2 );

PlotSeries( SMAMult, 0, 005, 2 );
DrawText( 'Detrended 30 Day SMA', 0, 4, 56, 005, 8 );
```

12.28 MultiplySeriesValue

MultiplySeriesValue(Series: integer; Value: float): integer;

ChartScripts SimuScripts PerfScripts CMScritps

Description

Multiplies the Price *Series* by a constant *Value*, and returns the integer handle to a new Price Series. You can use this function to make a copy of a Price Series by passing 1 as the *Value* parameter.

Example

```
{ Divide the stock value by the DJ Index to get a relative
  strength rating, multiple the result by 10000 for scaling }
var DJCLOSE, DIVSERIES, DJPANE: integer;
SetPrimarySeries( '^DJI' );
DJClose := #Close;
RestorePrimarySeries;
DivSeries := DivideSeries( #Close, DJClose );
DivSeries := MultiplySeriesValue( DivSeries, 10000 );
DJPane := CreatePane( 150, true, true );
PlotSeries( DivSeries, DJPane, 030, #Thick );
```

12.29 OffsetSeries

OffsetSeries(Series: integer; Bars: integer): integer;

ChartScripts SimuScripts PerfScripts CMScritps

Description

Offsets the specified Price Series by a certain number of Bars and returns the handle to a new Price Series. Use a negative offset value to shift the Price Series to the right on the chart. Shifting a moving average to the right often leads to cleaner signals.

Example

```
{ Display a 60 day moving average, and the same
  average shifted to the right by 4 bars }
var SMASER, SMAOFFSET: integer;
SMASer := SMAseries( #Close, 60 );
SMAOffset := OffsetSeries( SMASer, -4 );
PlotSeries( SMASer, 0, #Navy, #Thick );
PlotSeries( SMAOffset, 0, #Blue, #Thin );
```

12.30 RestorePrimarySeries

RestorePrimarySeries;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Restores the primary data source after a call to **SetPrimarySeries**. You should always call **RestorePrimarySeries** after accessing external data sources in your ChartScript.

Example

```
{ Compare the RSI of the stock with the RSI of the index }
var DJCLOSE, DJRSI, DJPANE: integer;
SetPrimarySeries( '^DJI' );
DJClose := #Close;
DJRSI := RSISeries( DJClose, 30 );
RestorePrimarySeries;
DJPane := CreatePane( 150, true, true );
PlotSeries( DJRSI, DJPane, #Navy, #Thick );
PlotSeries( RSISeries( #Close, 30 ), DJPane, #Blue, #Thin );
```

12.31 setDescription

setDescription(Series: integer; Description: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Every Price Series is associated with an internal description. You normally don't need to assign a description yourself, as this is usually done automatically.

setDescription assigns the *Description* string of the specified Price Series that will be returned by a call to **getDescription**. The description is displayed in the Data Window (**Ctrl+Alt+V**).

Remarks

- **setDescription** is primarily intended to provide a description for Price Series formed by **CreateSeries**.
- For custom indicators created from the result of another Price Series function such as **SubtractSeries**, **DivideSeries**, etc., use **setDescription** to change the Series' *internal description* after calculating the indicator. **FindNamedSeries** will then be

able to identify the *Series* by its *Description* string.

Example

```
var Bar, hSer: integer;
{ A description is automatically assigned }
hSer := CreateSeries;
Print( GetDescription( hSer ) );

{ Assign your own description }
SetDescription( hSer, 'MySeries1' );
Print( GetDescription( hSer ) );

{ Observe the description in the Data Window }
PlotSeries( hSer, 0, 0, 0 );
```

12.32 SetPrimarySeries

SetPrimarySeries(Symbol: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Use this function to temporarily set the default price data source to a different symbol. After calling this function, any reference to a standard Price Series, such as **#Open**, **#High**, **#Volume**, etc, or the Data Access functions such as **PriceClose** will use the data source specified by the *Symbol* parameter. You can also build indicators on the newly selected data source.

Remarks

- Wealth-Lab will first try to find the *Symbol* in the WatchList of the primary symbol, which is the one that was clicked. If it doesn't exist in the primary symbol's WatchList, it will try to find it in other WatchLists/DataSources, top to bottom, alphabetically.
- Use **AllowSymbolSearch** to limit symbol searches to specific WatchLists.
- After calling **SetPrimarySeries**, you can execute trades on other than the 'clicked' symbol. This feature is available in the ChartScript Window and EOD Scans (must select "Allow Complete Scan of Multi-Symbol Scripts") but will cause the \$imulator to terminate prematurely.
- When finished using the external data source, be sure to call **RestorePrimarySeries**.
- External Series functionality is not available for Tick or Second-based charts.

Example

```
{ I like to see the DJ Index and its 200 day moving
  average along with my chart }
var DJCLOSE, DJ200, DJPANE: integer;
SetPrimarySeries( '^DJI' );
DJClose := #Close;
DJ200 := SMAseries( DJClose, 200 );
RestorePrimarySeries;
DJPane := CreatePane( 150, true, true );
PlotSeries( DJClose, DJPane, 310, #Thick );
PlotSeries( DJ200, DJPane, #Black, #Dotted );
```

12.33 SetSeriesValue

SetSeriesValue(Bar: integer; Series: integer; Value: float);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Assigns a value to a specific *Bar* in the specified *Price Series*. Use this function to set values of a *Price Series* that you create with **CreateSeries** or **CreateNamedSeries**.

Remarks

Instead of using **SetSeriesValue**, you can use the @-symbol shorthand notation. In the example below, the statement

```
SetSeriesValue( Bar, VolSurge, pct );
```

is equivalent to

```
@VolSurge[ Bar ] := pct;
```

Note: The @ syntax is not compatible with *Price Series* whose handles are stored in a declared array; e.g., @h[i][Bar], where h[i] is an integer array of *Price Series* handles, is not valid syntax. See the *WealthScript Language Guide* for more information.

Example

```
{ Below we create and plot an indicator that displays
  percentage of Volume above average }
var V, DIFF, PCT: float;
var VOLPANE, VOLSURGE, BAR: integer;
VolPane := CreatePane( 100, false, true );
VolSurge := CreateSeries;
for Bar := 20 to BarCount - 1 do
begin
  v := Volume( Bar );
  diff := v - SMA( Bar, #Volume, 20 );
  pct := ( diff / SMA( Bar, #Volume, 20 ) ) * 100;
  SetSeriesValue( Bar, VolSurge, pct );
end;
PlotSeries( VolSurge, VolPane, 955, #ThickHist );
```

12.34 SingleCalcMode

SingleCalcMode(Mode: boolean);

ChartScripts SimuScripts PerfScripts CMScripts

Description

When the *Mode* parameter is set true, selected indicators (see Remarks) will return the value for the specified bar only.

Discussion

When you first call an indicator function, Wealth-Lab constructs the entire indicator *Price Series* to optimize speed for accessing indicator values. However, this makes it difficult to use indicators within a loop as you are populating the values of a custom *Price Series*. **SingleCalcMode** allows you to use selected indicator functions repetitively to recalculate a result based on the instantaneous values of the underlying series.

Remarks

- **SingleCalcMode** will work with a subset of supported indicators: **SMA, EMA, LinearReg, RSquared, StdDev, Highest, Lowest**
Check the Build descriptions on the Wealth-Lab site for the latest additions.
- Set **SingleCalcMode** *true* only for the lines of code where required. When not required, set the *Mode* back to *false*.
- Some indicators do not lend themselves to bar by bar calculation, because their values depend on the previous value, or they use other complex indicators as components. If you try to call one of these indicators when **SingleCalcMode** is turned on you will receive an error message as a notification that the script might not be working as intended.
- Note that a call to **SMASeries**, for example, will create the complete Price Series as normal, but you could still call **SMA** within **SingleCalcMode** to return a newly-calculated value for a specific bar.

Example

```
{ Instructions: Load Fixed 100 Bars and compare the results by
  changing the SingleCalcMode parameter to true/false }
var n, MySeries, Pane: integer;

MySeries := CreateSeries;
for n := 0 to 19 do
  @MySeries[n] := n;

{ Change the Mode value here to see the effect }
SingleCalcMode( false );

for n := 20 to BarCount - 1 do
  @MySeries[n] := SMA( n, MySeries, 20 );

SingleCalcMode( false );
Pane := CreatePane( 100, true, true );
PlotSeries( MySeries, Pane, #Red, #Thin );
```

12.35 SubtractSeries

SubtractSeries(Series1: integer; Series2: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Subtracts each element of *Series2* from its corresponding element in *Series1*, and returns the handle to a new Price [difference] Series.

Example

```
{ Visualize the Difference between two Rate of Changes }
var ROC1, ROC2, ROCDIFF, ROCPANE: integer;
ROC1 := ROCSeries( #Close, 10 );
ROC2 := ROCSeries( #Close, 30 );
ROCDiff := SubtractSeries( ROC2, ROC1 );
ROCPane := CreatePane( 100, true, true );
PlotSeries( ROCDiff, ROCPane, 012, #ThickHist );
PlotSeries( ROC1, ROCPane, #Red, #Thin );
PlotSeries( ROC2, ROCPane, #Blue, #Thin );
```

12.36 SubtractSeriesValue

SubtractSeriesValue(Series: integer; Value: float): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Subtracts a constant *Value* from each element of the specified *Price Series*, and returns the handle of a new *Price Series*.

Example

```
{ By subtracting 50 from the RSI we can get an
  indicator that oscillates around zero }
var RSISer, RSIPane: integer;
RSIPane := CreatePane( 75, true, true );
RSISer := RSISeries( #Close, 30 );
RSISer := SubtractSeriesValue( RSISer, 50 );
PlotSeries( RSISer, RSIPane, 009, #Thin );
```

12.37 SubtractValueSeries

SubtractValueSeries(Value: float; Series: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Subtracts each element of a *Price Series* from the specified constant *Value*, and returns the handle of a new *Price Series*.

Example

```
{ Create a mirror-image by flipping Williams %R }
var PctRPane: integer;
PctRPane := CreatePane( 75, true, true );
PlotSeries( WilliamsRSeries( 30 ), PctRPane, 520, #Thick );
PlotSeries( SubtractValueSeries( 100, WilliamsRSeries( 30 ) ),
PctRPane, 250, #Thick );
DrawLabel( 'WilliamsR(30)', PctRPane );
```

12.38 SynchAll

SynchAll;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Synchronizes any *Secondary data series* that were used in the script (see **SynchSeries**).

Example

```
{ The Oddball System requires synchronized data
  from the NYSE Advances, symbol $ADV }
var ADV, ADVPANE, ADROC, ROCPANE, BAR: integer;

{ Obtain the NYSE Advancing Issues }
ADV := GetExternalSeries( '$ADV', #Close );
SynchAll;
HideVolume;
```

```

{ Plot it }
ADVPane := CreatePane( 75, true, true );
PlotSeries( ADV, ADVPane, 112, #Thick );
DrawLabel( 'NYSE Advancing Issues', ADVPane );

{ Get the 7 bar ROC of the NYSE Advances }
ADROC := ROCSeries( ADV, 7 );

{ Plot it }
ROCPane := CreatePane( 75, true, true );
PlotSeries( ADROC, ROCPane, 224, #ThickHist );
DrawLabel( '7 bar ROC of NYSE Advancing Issues', ROCPane );

{ Oddball Trading Rules }
for Bar := 20 to BarCount - 1 do
begin
  if GetTime( Bar ) <= 1500 then
  begin
    { Currently in a long Position }
    if PositionShort( LastPosition ) then
    begin
      if ROC( Bar, ADV, 7 ) > 3 then
      begin
        CoverAtClose( Bar, LastPosition, '' );
        BuyAtClose( Bar, '' );
      end;
    end
  else
    { Currently in a short Position }
    begin
      if ROC( Bar, ADV, 7 ) < 1 then
      begin
        SellAtClose( Bar + 1, LastPosition, '' );
        ShortAtClose( Bar + 1, '' );
      end;
    end;
  end;
end;
end;
end;
{$I 'Profit Pane (Bottom)'}

```

12.39 SynchSeries

SynchSeries(Symbol: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Synchronizes the data in a Secondary data series from *Symbol* (obtained from **GetExternalSeries** or **SetPrimarySeries**) to the Primary data series, so that all bars are perfectly aligned by date. There are two actions that **SynchSeries** can perform on the Secondary Series (it never modifies the Primary Series). First, if the Secondary Series contains dates that do not exist in the Primary Series, these bars are eliminated from the Secondary Series. Next, if the Primary Series contains dates that do not exist in the Secondary, synthetic bars of data are added to the Secondary Series to make up the missing bars. These synthetic bars obtain their value from the previous bar's data.

You should call **SynchSeries** only after creating any desired indicators from the Secondary Series. The synchronization process is applied to all indicators created from the Secondary Series, ensuring that they align properly with the dates in the

Primary Series. If you create your indicators after calling **SynchSeries**, bars may be removed or duplicated, and the indicators would be based on imperfect data.

However, if you want to perform operations that combine the Secondary Series and the Primary (using **DivideSeries** for example), do so after the synchronization has taken place.

Example

```
{ This script shows visually how data might be misaligned.
  It obtains the Nasdaq index, then makes a copy of the data.
  The SynchSeries then synchronizes the original data series.
  The script then plots both the synchronized and the original
  Series so you can see if any data problems existed. }
var NAZ, NAZCOPY, NAZPANE: integer;
Naz := GetExternalSeries( '^IXIC', #Close );
NazCopy := MultiplySeriesValue( Naz, 1 );
SynchSeries( '^IXIC' );
NazPane := CreatePane( 100, true, true );
PlotSeries( Naz, NazPane, #Black, #Thin );
PlotSeries( NazCopy, NazPane, #Red, #Thin );
```

12.40 SyntheticBar

SyntheticBar(Bar: integer): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true if the specified *Bar* number is a "synthetic" bar. A synthetic bar is a bar that was added to the chart as a result of a call of **AddCalendarDays** or **AddFutureBars**.

12.41 TurnDown

TurnDown(Bar: integer; Series: integer): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true if the *Price Series* has turned down as of the specified *Bar*. A *Price Series* has "turned down" if its value is less than its previous value, and its previous value was greater than or equal to the value preceding it.

Specifically, given a *Series D*, **TurnDown** returns true if

```
@D[Bar] < @D[Bar - 1], and,
@D[Bar - 1] >= @D[Bar - 2]
```

Example

```
{ Buy when Williams %R turns down and is above 80 }
var PctRPane: integer;
PctRPane := CreatePane( 75, true, true );
PlotSeries( WilliamsRSeries( 30 ), PctRPane, 009, #Thin );
var Bar: integer;
InstallStopLoss( 5 );
InstallProfitTarget( 10 );
for Bar := 30 to BarCount - 1 do
begin
```

```

ApplyAutoStops( Bar );
if TurnDown( Bar, WilliamsRSeries( 30 ) ) then
  if WilliamsR( Bar, 30 ) > 80 then
    BuyAtMarket( Bar + 1, 'WR' );
  end;
end;

```

12.42 TurnUp

TurnUp(Bar: integer; Series: integer): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true if the Price *Series* has turned up as of the specified *Bar*. A Price Series "turns up" if its value is greater than its previous value, and its previous value was less than or equal to the value preceding it.

Specifically, given a Series *U*, **TurnUp** returns true if

```

@U[Bar] > @U[Bar - 1], and,
@U[Bar - 1] <= @U[Bar - 2]

```

Example

```

{ Enter the market when the slow stochastic turns up from below 15 }
var Bar, StochPane: integer;
StochPane := CreatePane( 100, true, true );
PlotSeries( StochDSeries( 60, 5 ), StochPane, 202, #Thick );
for Bar := 65 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    if StochD( Bar - 1, 60, 5 ) < 15 then
      if TurnUp( Bar, StochDSeries( 60, 5 ) ) then
        BuyAtMarket( Bar + 1, 'StochasticD Turns Up' );
      end
    else
    begin
      if CrossOverValue( Bar, StochDSeries( 60, 5 ), 80 ) then
        SellAtMarket( Bar + 1, LastPosition, 'StochD Crosses 80' );
      end;
    end;
  end;
end;

```


13 SimuScript Functions

13.1 Overview

SimuScripts are an advanced feature of Wealth-Lab Developer 4.0 that let you experiment with your very own position-sizing rules in the \$imulator as well as in the ChartScript, Rankings, and Scans tools when Portfolio Simulation mode is selected. A SimuScript is a special type of ChartScript that must be stored in the "SimuScripts" folder. For more information, see the chapter on SimuScripts in the WealthScript Function Reference.

Generally speaking, besides the specific SimuScript-category functions, you can use any of the functions in the other categories **excluding** the functions that appear in the following categories:

- Alerts
- Cosmetic Charts
- System
- Time Frame
- Trading System

13.2 BarCount

BarCount: integer;

ChartScripts *SimuScripts PerfScripts CMScripts

Description

Returns the total number of bars available at the time the current \$imulator (or Portfolio Simulation) trade was opened.

Example

```
{ See if we've had at least a 20% Equity rise in the last 100 Bars }
var CHANGE: float;
var GAIN20: boolean;
Gain20 := false;
if BarCount > 100 then
begin
  Change := Equity( BarCount - 1 ) - Equity( BarCount - 100 );
  Change := ( Change / Equity( BarCount - 100 ) ) * 100;
  if Change >= 20 then
    SetPositionSizePct( 50 )
  else
    SetPositionSizePct( 5 );
end;
```

13.3 BuyAndHold

BuyAndHold(Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Buy & Hold Equity curve value at the specified *Bar*.

Example

```

{ Use any surges in the Buy & Hold equity curve to
  increase out position size }
var xCurrent, xPast, xDiff, xPct: float;
SetPositionSizePct( 10 );
if BarCount > 20 then
begin
  xCurrent := BuyAndHold( BarCount - 1 );
  xPast := BuyAndHold( BarCount - 20 );
  xDiff := xCurrent - xPast;
  xPct := ( xDiff / xCurrent ) * 100;
  if xPct > 0 then
    SetPositionSizePct( 10 + xPct );
end;

```

13.4 CandidateCount

CandidateCount: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the number of *potential* Positions that are being processed for the bar on which the SimuScript was called.

Remarks

- **CandidateCount** can be used, for example, to spread current cash or equity equally over trading signals generated on the same bar in Portfolio \$imulations only (see warning).

Warning! Generally, use of **CandidateCount** should be restricted to testing trading systems that use market order entries exclusively. If used in conjunction with stop or limit entries a *peeking effect can occur* since only the theoretical trades that actually take place are counted.

- **CandidateCount** returns the count of all *Alerts* for all order types (Market/Limit/Stop/Close).

Example

```

{ Spread free cash equally over multiple signals generated on the same
  bar if Cash > 10000. (No margin assumed) }
const MINSIZE = 10000;
var Bar: integer = BarCount - 1;
var Size: float;

if Cash( Bar ) < MINSIZE then
  SetPositionSizeShares( 0 )
else if CandidateCount = 1 then
  SetPositionSizeFixed( MINSIZE )
else
begin
  Size := Cash( Bar ) / CandidateCount;
  SetPositionSizeFixed( Size );
end;

```

13.5 Cash

Cash(Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Cash level at the specified *Bar*. The Cash level is equal to Equity minus current market Exposure.

Warning! The **Cash** function is also available for ChartScripts, but it cannot be employed in a system's trading rules if destined for the \$imulator.

Example

```
{ SimuScript: Don't take a new Position if our Cash reserve is less
than $5,000.
  If this happens we need a vacation anyway. }
if Cash( BarCount - 1 ) < 5000 then
  SetPositionSizeShares( 0 );
```

13.6 DrawDown

DrawDown(Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the drawdown for the specified *Bar*. **DrawDown** is calculated on a closing basis and is the dollar amount *below* the last peak in the equity curve.

Remarks:

- **DrawDown** retains its negative sign. For example, a drawdown of \$5,000 is returned as -5000.
- To obtain the **DrawDown** for the current bar in a SimuScript, pass **BarCount - 1** as the *Bar* parameter.

Warning! **DrawDown** is also available for ChartScripts, but it cannot be employed in a system's trading rules if destined for the \$imulator.

Example

```
{ SimuScript sizing example based on drawdown }
var HighestEquity: float;

{ Use Global Storage to track Highest Equity }
if PositionCount = 0 then
begin
  ShowMessage( 'Init' );
  SetGlobal( 'Highest', Equity( 0 ) );
end;

HighestEquity := GetGlobal( 'Highest' );
if Equity( BarCount - 1 ) > HighestEquity then
begin
  HighestEquity := Equity( BarCount - 1 );
  SetGlobal( 'Highest', HighestEquity );
end;
```

```

{ Have we doubled account size? }
if HighestEquity > Equity( 0 ) * 2 then
begin
  if -DrawDown( BarCount - 1 ) > HighestEquity * 0.10 then
    SetPositionSizePct( 0 )
  else
    SetPositionSizePct( 10 );
end
else
  SetPositionSizePct( 25 );

```

13.7 DrawDownPct

DrawDownPct(Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the percentage drawdown for the specified *Bar*. **DrawDownPct** is calculated on a closing basis and represents the percent decrease in equity since the last peak in the equity curve.

Remarks:

- **DrawDownPct** is reported in percentage terms, and, it retains its negative sign. For example, a 5% drawdown is returned as -5.
- To obtain the **DrawDownPct** for the current bar in a SimuScript, pass **BarCount - 1** as the *Bar* parameter.

Warning! **DrawDownPct** is also available for ChartScripts, but it cannot be employed in a system's trading rules if destined for the \$imulator.

Example

```

{ SimuScript: Preserve Capital by taking no trades during large
drawdown periods }
if DrawDownPct( BarCount - 1 ) < -5 then
  SetPositionSizeShares( 0 )
else
  SetPositionSizePct( 10 );

```

13.8 Equity

Equity(Bar: integer): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the total account value at the specified *Bar*. In Wealth-Lab, **Equity** refers to the total Cash plus Market Exposure.

Remarks

- Do not use **Equity** in a ChartScript that is destined for the \$imulator, which processes *Portfolio Equity* while sizing Positions following ChartScript execution. Search for the following articles in the Knowledge Base for more information: *Interacting Dynamically with Portfolio Level Equity* and *Understanding the \$imulator*.

Warning! The **Equity** function is also available for ChartScripts, but it cannot be employed in a system's trading rules if destined for the \$imulator.

Example

```
{ SimuScript
  Take the money and run! }
if Equity( BarCount - 1 ) > 120000 then
  SetPositionSizeShares( 0 );
```

13.9 SetPositionSizeFixed

SetPositionSizeFixed(Value: float);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Instructs the \$imulator (or Portfolio Simulation) to assign a fixed Position size, a dollar *Value*, to the Position currently being processed by the SimuScript.

Example

```
{ Set a Position size of half current available cash }
SetPositionSizeFixed( Cash( BarCount - 1 ) / 2 );
```

13.10 SetPositionSizePct

SetPositionSizePct(Value: float);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Instructs the \$imulator (or Portfolio Simulation) to assign a percentage, *Value*, of the total Equity size (the net value of the Portfolio, cash + equities) to the Position that is currently being processed by the SimuScript.

Example

```
{ The ChartScript Code used SetPositionData to establish
  the percentage of equity that should be used. The
  "PositionData" must be set in the ChartScript code. }
x := GetPositionData( #Current );
SetPositionSizePct( x );
```

13.11 SetPositionSizeShares

SetPositionSizeShares(Shares: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Tells the \$imulator (or Portfolio Simulation) to assign a fixed number of *Shares* to the Position currently being processed.

Example

```
{ Set the value number of shares based on the position's basis price. }
var Basis: float;
var Shrs: integer;

Basis := PositionBasisPrice( #Current );
if Basis < 10 then
  Shrs := 800
else if Basis < 20 then
  Shrs := 500
else if Basis < 50 then
  Shrs := 300
else
  Shrs := 200;

SetPositionSizeShares( Shrs );
```

13.12 SortByEntryDate

SortByEntryDate;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Instructs the SimuScript to access Positions in the order in which they were opened. The sort order becomes important if you write SimuScripts that assign Position size based on streaks.

Remarks

\$imulator only. Not available for Portfolio Simulations ran from the ChartScript window.

Example

```
{ Increase Position Size based on last 5 submitted trades }
var n, p: integer;
var x: float;
x := 1000;
n := 0;
SortByEntryDate;
for p := PositionCount - 1 downto 0 do
  if not PositionActive( p ) then
    begin
      n := n + 1;
      if n > 5 then
        Break;
      if PositionProfit( p ) > 0 then
        x := x * 2;
    end;
  SetPositionSizeFixed( x );
```

13.13 SortByExitDate

SortByExitDate;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Instructs the SimuScript to access Positions in the order in which they were closed. The sort order becomes important if you write SimuScripts that assign Position size based on streaks.

Remarks

Simulator only. Not available for Portfolio Simulations ran from the ChartScript window.

Example

```
{ Increase Position Size as we get streaks of winners }
var p: integer;
var x: float;
x := 1000;
SortByExitDate;
for p := PositionCount - 1 downto 0 do
  if not PositionActive( p ) then
    if PositionProfit( p ) > 0 then
      x := x + 1000
    else
      Break;
SetPositionSizeFixed( x );
```

14 String Functions

14.1 Overview

To compare, parse, or otherwise manipulate string variables, the String category of functions has what it takes.

14.2 CharAt

CharAt(Value: string; Index: integer): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns a single string character at the *Index* position from the *Value* string.

Remarks

- The *Index* of the first character in the *Value* string is 1.
- If the *Index* is invalid, CharAt returns "NUL", i.e., ASCII Code 0.
- To return a complete substring instead of just one character, use **Copy**.

Example

```
{ A re-useable vertical labeling routine }
procedure VerticalLabel( Bar: integer; str: string; AbovePrices:
boolean; Color, Size: integer );
begin
var n: integer;
    if AbovePrices then
    begin
        for n := Length( str ) downto 1 do
            AnnotateBar( CharAt( str, n ), Bar, AbovePrices, Color, Size );
    end
    else
        for n := 1 to Length( str ) do
            AnnotateBar( CharAt( str, n ), Bar, AbovePrices, Color, Size );
    end;

const S1 = 'PEAK';
const S2 = 'TROUGH';
var Bar, PB, TB: integer;

{ Make extra room for peak label }
HidePaneLines;
CreatePane( 20, true, false );

Bar := BarCount - 1;
PB := PeakBar( Bar, #Close, 5 );
VerticalLabel( PB, S1, true, #Blue, 8 );
TB := TroughBar( Bar, #Close, 5 );
VerticalLabel( TB, S2, false, #Red, 8 );
```


14.3 Chr

Chr(ASCIICode: integer): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the string character of the specified *ASCIICode*, which may be a literal whole number or integer variable.

Remarks

- Alternatively, you may use the shorthand "#*ascii*code" notation. In this case, you must use a literal number from 0 to 255, inclusive, as shown in the example.
- See also: **Ord**

Example

```
{ Print a list of the ASCII characters to the debug
  window starting with printable characters }
var i: integer;
for i := 33 to 255 do
{ Separate the items by a Tab character, ASCII 9 }
  Print( IntToStr( i ) + #9 + Chr( i ) );
```

14.4 CompareStr

CompareStr(s1: string; s2: string): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Compares two strings, *s1* and *s2*, case sensitively. The function returns 0 if the two strings are equal. If the first string is greater than the second (alphabetically), the function returns a positive integer, and it returns a negative integer if the second string is greater.

Example

```
var s1, s2: string;
s1 := Input( 'Enter First String' );
s2 := Input( 'Enter Second String' );
ShowMessage( 'CompareStr Result is: '
             + IntToStr( CompareStr( s1, s2 ) ) );
```

14.5 CompareText

CompareText(s1: string; s2: string): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Compares two strings, *s1* and *s2*, without case sensitivity. The function returns 0 if the two strings are equal. If the first string is greater than the second (alphabetically), the function returns a positive integer, and it returns a negative integer if the second string is greater.

Example

```

var s1, s2: string;
s1 := Input( 'Enter First String' );
s2 := Input( 'Enter Second String' );
ShowMessage( 'CompareText Result is: '
             + IntToStr( CompareText( s1, s2 ) ) );

```

14.6 Copy

Copy(String: string; Index: integer; Count: Integer): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns a substring from the specified *String*. The substring begins at the position specified in the *Index* parameter, and has a length specified by the *Count* parameter.

Example

```

function GetFirstWord( s: string ): string;
var n: integer;
begin
  Result := '';
  n := Pos( ' ', s );
  if n > 2 then
    Result := Copy( s, 1, n - 1 );
end;
DrawLabel( GetFirstWord( GetSecurityName ), 0 );

```

14.7 Delete

Delete(String: string; Index: integer; Count: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Removes a substring from the specified *String*. The substring that is removed begins at the value of the *Index* parameter, and has a length specified by the *Count* parameter.

Example

```

var s: string;
s := 'Honest Abe Lincoln';
Delete( s, 8, 4 );
ShowMessage( s );

```

14.8 FloatToStr

FloatToStr(Value: float): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Converts the specified floating point *Value* to a string. For more control over the conversion process, use the **FormatFloat** function.

Example

```

{ Display the change over a one year period }

```

```

var X, X2, XCHANGE, XPCT: float;
x := PriceClose( BarCount - 251 );
x2 := PriceClose( BarCount - 1 );
xChange := x2 - x;
xPct := ( xChange / x ) * 100;
DrawLabel( 'Net Change over a past year: '
          + FloatToStr( xPct ) + '%', 0 );

```

14.9 FormatFloat

FormatFloat(FormatString: string; Value: float);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Converts the specified floating point value to a string, using the specified *FormatString*. The following characters most commonly appear in the format string:

- 0** specifies a digit and forces leading/trailing zeroes if *Value* does not have a digit that falls in the position appearing in *FormatString*
- #** same as 0, but displays nothing if *Value* has no corresponding digit
- ,** (comma) digit grouping symbol (thousands only)
- .** (period) decimal point
- other* Other printable characters are displayed as literals in the position where they appear in the format string. You can add currency symbols (\$, €, etc.) or the percent sign (%), e.g., '\$#,##0.00' or '0.0%'

Remarks

- Using a comma "," in *FormatString* will add the *Digit grouping symbol* defined in your Windows Regional Options Numbers tab. Regardless of the comma's position in the format string, the grouping will be by *thousands*.
- Using a period "." in the format code will add a *Decimal symbol* defined in your Windows Regional Options Numbers tab. Using more than one period in *FormatString* will generate an error.

Example

```

{ Print a closing value to the debug window }
var BAR: integer;
Print( FormatFloat( '#,##0.00', PriceClose( Bar ) ) );

```

14.10 GetToken

GetToken(String: string; TokenNum: integer; Delimiter: string): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Parses a *String* and returns an individual token. The *String* itself remains unaffected by the function call. You specify a character to use as a *Delimiter*, and the token number, *TokenNum*, to return. Pass 0 to return the first token, 1 for the second, etc.

Remarks

- **GetToken** returns a null string (ASCII Code 0) if the specified *TokenNum* does not

exist in the *String*.

Example

```
{ Return a token from a space-delimited string }
s := 'This is a line of tokens';
sToken := GetToken( s, 5, ' ' );
{ sToken now contains the string "tokens" }
```

14.11 Insert

Insert(Source: string; S: string; Index: Integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Inserts a substring, *Source*, into a string *S* beginning at a point specified by the *Index* parameter.

Example

```
var s: string;
s := 'Honest Lincoln';
Insert( 'Abe ', s, 8 );
ShowMessage( s );
```

14.12 IntToStr

IntToStr(Value: integer): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Convert the integer value specified in the *Value* parameter to a string.

Example

```
{ Display the total number of bars in the chart }
DrawLabel( 'The Chart has ' + IntToStr( BarCount ) + ' Bars', 0 );
```

14.13 Length

Length(String: string): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the length of the specified *String*.

Example

```
var s: string;
s := 'Wealth-Lab';
n := Length( s ); //n is now 10
```

14.14 LowerCase

LowerCase(Value: string): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns a lowercase copy of the specified string, *Value*.

Example

```
{ Capitalize the first letter of the name only }
var S, FIRSTLETTER: string;
s := LowerCase( GetSecurityName );
if s = '' then
    Exit;
FirstLetter := UpperCase( Copy( s, 1, 1 ) );
s := FirstLetter + Copy( s, 2, Length( s ) );
ShowMessage( s );
```

14.15 Ord

Ord(Value: string): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the ordinality (ASCII code) of the first character of the specified *Value*.

Remarks

- *Value* can be any sequence of characters, but must not be a blank string.
- See also: **Chr**

Example

```
{ Create a table of printable characters }
var c: integer;
for c := 32 to 128 do
    Print( IntToStr( c ) + #9 + Chr( c )
          + #9 + IntToStr( Ord( Chr( c ) ) ) );
```

14.16 Pos

Pos(SubString: string; String: string): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the index value (position) of the first character in a specified *Substring* that occurs in a given *String*.

Example

```
var s: string;
var n: integer;
s := Input( 'Give me a string!' );
n := Pos( 'A', UpperCase( s ) );
if n = 0 then
    ShowMessage( 'The letter 'A' is not present.' );
```

```

else
  ShowMessage( 'The letter 'A' is at position ' + IntToStr( n ) + '.'
);

```

14.17 StrToFloat

StrToFloat(Value: string): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Converts the specified string, *Value*, to a floating point value.

Example

```

{ Read a value from an external file and convert to floating point }
var f: integer;
var s: string;
var x: float;
f := FileOpen( 'MyFile.txt' );
s := FileRead;
x := StrToFloat( s );

```

14.18 StrToFloatDef

StrToFloatDef(Value: string; Default: float): float;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Converts the specified string, *Value*, to a floating point value. If the string does not represent a valid floating point value, the function returns the value provided in the *Default* parameter.

Example

```

{ Let the user specify the Standard Deviations to use for Bollinger
Bands }
var S: string;
var X: float;
s := Input( 'Bollinger Band Std Dev?' );
x := StrToFloatDef( s, 1.5 );
PlotSeries( BBandLowerSeries( #Close, 10, x ), 0, 009, #Thin );
PlotSeries( BBandUpperSeries( #Close, 5, x ), 0, 009, #Thin );

```

14.19 StrToInt

StrToInt(Value: string): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Converts the string specified in the *Value* parameter to an integer.

Example

```

function ConvertDate( Date: string ): integer;
begin
  var y, m, d: string;

```

```

m := GetToken( Date, 0, '/' );
d := GetToken( Date, 1, '/' );
y := GetToken( Date, 2, '/' );

Result := StrToInt( y ) * 10000
        + StrToInt( m ) * 100
        + StrToInt( d );
end;

const MyDate = '5/18/2005';
Print( 'ConvertDate returns: ' + IntToStr( ConvertDate( MyDate ) ) );
try
  Print( 'StrToDate returns: ' + IntToStr( StrToDate( MyDate ) ) );
except
  Print( '' );
  Print( 'Your computer's short date format is not mm/dd/yyyy' );
end;

```

14.20 StrToIntDef

StrToIntDef(Value: string; Default: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Converts the string specified in the *Value* parameter to an integer. If the string isn't a valid integer, the function instead returns the value in the *Default* parameter.

Example

```

{ Let the user specify the period they want to use }
var S: string;
var N: integer;
s := Input( 'EMA Period?' );
n := StrToIntDef( s, 50 );
PlotSeries( EMASeries( #Close, n ), 0, 002, #Thick );

```

14.21 Trim

Trim(s: string): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns a copy of the specified string, *s*, with leading and trailing blanks trimmed off.

Example

```

{ Read a line from an external file }
var s: string;
var n: integer;
n := FileOpen( 'C:\MyFile.txt' );
s := FileRead( n );
s := Trim( s );

```

14.22 TrimLeft

TrimLeft(s: string): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns a copy of the specified string, *s*, with leading blanks trimmed off.

Example

```
{ Get second word of company name }
var s: string;
var n: integer;
s := GetSecurityName;
n := Pos( ' ', s );
if n > 0 then
begin
  s := Copy( s, n, Length( s ) );
  s := TrimLeft( s );
end;
```

14.23 TrimRight

TrimRight(s: string): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns a copy of the specified string, *s*, with trailing blanks trimmed off.

Example

```
{ Output contents of a file }
var s: string;
var f: integer;
f := FileOpen( 'C:\MyFile.txt' );
while not FileEOF( f ) do
  Print( TrimRight( FileRead( f ) ) );
```

14.24 UpperCase

UpperCase(s: string): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns an uppercase copy of the specified string, specified in the parameter *s*.

Example

```
{ Print the names of all your ChartBooks to the Debug window }
var F: integer;
f := FileOpen( 'Namespaces.txt' );
while not FileEOF( f ) do
  Print( UpperCase( FileRead( f ) ) );
```


15 System Functions

15.1 Overview

In the System category, you'll find a broad array of functions whose primary purpose is to interact with the user and external objects or programs. They also provide a manner to control and generate output for Scans, the Commentary Window, and chart image files. Finally, a subset of the System Functions furnish methods to easily work with an entire group of symbols - a WatchList.

Note: Generally speaking, the System category of WealthScript functions are not available for SimuScripts.

15.2 Abort

Abort;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Causes a script to stop processing immediately. Wealth-Lab reports an error message upon executing an **Abort** statement.

Remarks

- You can abort ChartScript processing manually by striking the **Esc** key.
- Use **Exit** instead of **Abort** to terminate/exit a procedure without error.

Example

```
if BarCount < 100 then
  Abort;
```

15.3 AddCommentary

AddCommentary(Line: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Adds a *Line* of commentary to the ChartScript Commentary window. You can use any valid HTML tags in your commentary (see the User Guide in the Help File for some examples). To force a line break end your string with the tag '
'.
</div>

Tip: To open the Commentary window, select **View|Commentary Window (Ctrl+Alt+C)** from the main menu, or select its icon from the View toolbar.

Example

```
var Bar: integer;
var s: string;
var x1, x2, x: float;
const FMT = '#0.00';
```

```

Bar := BarCount - 1;
s := '<h1>' + GetSymbol + '</h1>';
AddCommentary( s );
if not ( GetSecurityName = '' ) then
begin
  s := '<h2>(' + GetSecurityName + ')</h2>';
  AddCommentary( s );
end;
AddCommentary( '<b>Dual Stochastic Sell Strategy</b><br>' );
x1 := StochD( Bar, 45, 5 );
s := 'FastD(45): ' + FormatFloat( FMT, x1 ) + '<br>';
AddCommentary( s );
x2 := StochD( Bar, 7, 5 );
s := 'FastD(7): ' + FormatFloat( FMT, x2 ) + '<br>';
AddCommentary( s );
x1 := Lowest( Bar, #Low, 10 );
s := 'Lowest 10 Bar Low: $' + FormatFloat( FMT, x1 ) + '<br>';
AddCommentary( s );

```

15.4 AddScanColumn

AddScanColumn(Name: string; Value: float);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Adds a custom column, *Name*, to the Real-Time Scans and WatchList Scans tools, and populates the column with the numeric value specified in the *Value* parameter. You can sort the symbols in the Scan tools by clicking on the heading of your custom columns.

Remarks

- The *Value* displayed in the custom column is fixed to 2 decimals of significance. Use **AddScanColumnStr** to display more precision.
- Typically, you'll want to provide the value of one or more indicators on the final bar of the chart, i.e. the signal bar. The data is displayed in the Custom Columns View after running a WatchList Scan, or as a new column in the Scan Results for Real-Time Scans.
- **AddScanColumn** should not be used conditionally. Since all scanned symbols are always added to the Custom Columns view of the Scans tool, adding values conditionally would create variable length records, which can cause printing incompatibilities.

Example

```

{ Add the most recent MACD as a custom column in the WatchList Scan tool }
var BAR: integer;
Bar := BarCount - 1;
AddScanColumn( 'MACD', MACD( Bar, #Close ) );

```

15.5 AddScanColumnStr

AddScanColumnStr(Name: string; Value: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Adds a custom column, *NAddame*, to the Real-Time Scans and WatchList Scans tools, and populates the column with the string specified in the *Value* parameter. You can sort the symbols in the Scan tools by clicking on the heading of your custom columns.

Remarks

- Use **AddScanColumnStr** with **FormatFloat** to control the display precision of a number.
- Typically, you'll want to provide a string message on the final bar of the chart, i.e. the signal bar. The data is displayed in the Custom Columns View after running a WatchList Scan, or as a new column in the Scan Results for Real-Time Scans.
- **AddScanColumnStr** should not be used conditionally. Since all scanned symbols are always added to the Custom Columns view of the Scans tool, adding values conditionally would create variable length records, which can cause printing incompatibilities.

Example

```
var C: float;
var S: string;
s := WatchListName;
AddScanColumnStr( 'List', s );

{ Show the final closing price with 3 decimals of precision }
C := PriceClose( BarCount - 1 );
AddScanColumnStr( 'Close', FormatFloat( '0.000', C ) );
```

15.6 AllowSymbolSearch

AllowSymbolSearch(DataSource: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Specifies the *DataSource(s)* that will be searched when an external symbol is requested that is not within the same *DataSource* as the symbol being executed in the chart. If you do not call **AllowSymbolSearch**, Wealth-Lab searches for the external data in all of your *DataSources* in alphabetical order. Call **AllowSymbolSearch** one or more times in the script to allow only certain *DataSources* to be searched.

Example

```
{ Allow the script to search these daily DataSources only and ignore
  Intraday DataSources }
AllowSymbolSearch( 'Dow 30' );
AllowSymbolSearch( 'Nasdaq 100' );
```

15.7 CreateOleObject

CreateOleObject(ClassName: string): ComVariant;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Creates an instance of an OLE (Object/Linking and Embedding) object with the specified COM *ClassName*. The return value of this function should be stored into a variable of data type **ComVariant**. Once created, you can call any methods that are available in the object.

Example

```
{ Interfacing to a Wealth-Lab add-in written as a COM DLL }
var lib: ComVariant;
var CustomSeries: integer;
lib := CreateOleObject( 'AddOnLib.WLAddOn' );
lib.ExecuteProc( 123, IWealthLabAuto );
CustomSeries := CreateSeries;
lib.CustomIndicator( CustomSeries, #Close, 24, IWealthLabAuto );
PlotSeries( CustomSeries, 0, #Red, #Thick );
```

15.8 GetGlobal

GetGlobal(VariableName: string): Variant;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Retrieves a variant value from the variable *VariableName* in a global storage area, which is available to all Wealth-Lab scripts, excluding PerfScripts. Assign values to the global storage area by using **SetGlobal**.

Values assigned to the global storage area retain their values between script runs. You can, for example, set a global variable in a ChartScript and then access the value in a SimuScript.

Remarks

- If the global variable *VariableName* does not exist **GetGlobal** returns **Null**.

Note: *In prior versions of Wealth-Lab, GetGlobal returned 0 for non-existent global variables. This made it difficult to store strings in a global variables and test for their existence.*

Example

```
{ Retrieve the BarCount stored in Global Storage. See SetGlobal example }
var MyVar: string;
if GetGlobal( 'AABarCount' ) = Null then
begin
  MyVar := Input( 'Enter a Value:' );
  SetGlobal( 'AABarCount', MyVar );
end
else
  MyVar := GetGlobal( 'AABarCount' );
ShowMessage( 'The value is: ' + MyVar );
```

15.9 GetScriptName

GetScriptName: string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the name of the ChartScript that is currently being executed.

Remarks

- In a SimuScript, **GetScriptName** returns the name of the *ChartScript*.
- Not valid for SimuScripts used by the \$imulator.

Example

```
var s: string;
s := GetScriptName;
if s = '' then
  s := 'Untitled';
Print( 'The ChartScript name is ' + s );
```

15.10 GetTickCount

GetTickCount: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the number of milliseconds that the operating system has been operating. This value is used for benchmarking purposes. By taking the TickCount before and after an operation, and then subtracting the difference, you can gauge how long an operation is taking to complete.

Example

```
var Bar, n1, n2, r, i: integer;
var x: float;
n1 := GetTickCount;
x := 0;
for i := 30 to BarCount - 1 do
  x := x + RSI( Bar, #Close, 30 );
n2 := GetTickCount;
ShowMessage( 'Loop took ' + IntToStr( n2 - n1 ) + ' ms to complete' );
```

15.11 Input

Input(Caption: string): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Displays an input dialog where the user can enter a value, which is always returned as a **string** type, that is returned by the function. The specified *Caption* is displayed as a message to prompt the user for input.

Note: The **Input** function is not compatible with ChartScript Integrated Debugger when *Stepping*.

Example

```
{ Get the period for an indicator each time the ChartScript is executed
}
var Period: integer;
var s: string;
```

```
s := Input( 'Indicator Period?' );
Period := StrToInt( s );
PlotSeriesLabel( EMASeries( #Close, Period ), 0, #Blue, #Thick, 'EMA('
+ s + ')' );
```

15.12 IWealthLabAddOn3

IWealthLabAddOn3: COMVariant;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns an instance of Wealth-Lab Developer 4.0's COM Add-On Interface. You can write add-ins to Wealth-Lab Developer 4.0 in any language capable of producing an ActiveX (COM) DLL. Your add-ins should accept the **IWealthLabAddOn3** interface as a parameter to their function calls. They can then utilize the methods of the interface to execute trades and create custom indicators.

For more information on the **IWealthLabAddOn3** interface see the *Add-On API* article on the Wealth-Lab.com web site.

Example

```
{ Interfacing to a Wealth-Lab add-in written in Visual Basic }
var lib: ComVariant;
var CustomSeries: integer;
lib := CreateOleObject( 'VBWL.VBWLInterface' );
lib.Execute( IWealthLabAddOn3 );
CustomSeries := CreateSeries;
lib.CustomIndicator( #Close, CustomSeries, 24, IWealthLabAddOn3 );
PlotSeries( CustomSeries, 0, #Navy, #Thick );
```

15.13 IWealthLabAuto

IWealthLabAuto: COMVariant;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns an instance of Wealth-Lab Developer 2.1's COM Automation Interface. You can write add-ins to Wealth-Lab Developer 2.1 in any language capable of producing an ActiveX (COM) DLL. Your add-ins should accept the **IWealthLabAuto** interface as a parameter to their function calls. They can then utilize the methods of the interface to execute trades and create custom indicators.

For more information on the **IWealthLabAuto** interface see the *COM Reference* article on the Wealth-Lab.com web site.

Example

```
{ Interfacing to a Wealth-Lab 2.1 add-in written in Visual Basic }
var lib: ComVariant;
var CustomSeries: integer;
lib := CreateOleObject( 'VBWL.VBWLInterface' );
lib.Execute( IWealthLabAuto );
CustomSeries := CreateSeries;
lib.CustomIndicator( #Close, CustomSeries, 24, IWealthLabAuto );
PlotSeries( CustomSeries, 0, #Navy, #Thick );
```

15.14 IsRealTime

IsRealTime: boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns boolean *true* if the script is accessing data from a Live Feed. Otherwise the function returns *false*.

Example

```
{ For example, use IsRealTime to disable LastBar logic }
var Bar: integer;
for Bar := 20 to BarCount - 1 do
begin
  if LastPositionActive then
  begin
    if LastBar( Bar ) and ( not IsRealTime ) then
      SellAtClose( Bar, LastPosition, 'EOD' );
    end
  else
  begin
    { Entry logic }
  end;
end;
```

15.15 Null

Null: variant;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns a Variant "Null" value. See **GetGlobal** for more information.

15.16 PlaySound

PlaySound(FileName: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Plays the sound specified in the *FileName* parameter. *FileName* should be a fully-qualified wave file (*.WAV).

For intraday trading systems that hold stop and/or limit exit signals open for multiple bars, you may not want to hear an audible Alert for each new bar. Below, we demonstrate how to play a sound for the entry signal and once only for the bracketed-order exit signal(s).

Example

```
{ Be sure to turn off 'Alert Triggered from ChartScript Window' in
Tools|Options|Sounds }
var Bar, p: integer;
var EntryPrice: float;
const EntrySound = 'C:\Program Files\Wealth-Lab, Inc\Wealth-Lab
Developer 3.0\Alert4.wav';
```

```

const ExitSound = 'C:\Program Files\Wealth-Lab, Inc\Wealth-Lab
Developer 3.0\Alert1.wav';

PlotStops;
for Bar := 20 to BarCount - 1 do
begin
  if LastPositionActive then
  begin
    p := LastPosition;
    EntryPrice := PositionEntryPrice( p );

    { Sound exit alert only if the first bar after entry is the last bar
in the chart }
    if Bar = BarCount - 1 then
      if Bar - PositionEntryBar( p ) = 0 then
        PlaySound( ExitSound );

        if not SellAtStop( Bar + 1, EntryPrice * 0.95, p, '5% StopLoss' )
then
          SellAtLimit( Bar + 1, EntryPrice * 1.08, p, '8% ProfitTarget' )
        end
      else
        if TurnUp( Bar, SMASeries( #Close, 20 ) ) then
        begin
          BuyAtMarket( Bar + 1, '' );
          if Bar = BarCount - 1 then // Sound entry alert
            PlaySound( EntrySound );
          end;
        end;
      end;
end;
end;

```

15.17 Print

Print(Value: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Prints the specified string *Value* into the Debug window.

Remarks

- All messages are printed to the Debug window upon completion of script processing. Use **PrintFlush** to immediately force debug strings to appear in the Debug window.
- The Debug window is cleared at the end of ChartScript processing prior to printing messages from the queue. Consequently, the window displays only the results from the *most recent* ChartScript run.

Example

```

{ Print the bars where there were SMA crossovers }
if CrossOver( Bar, SMASeries( #Close, 20 ), SMASeries( #Close, 60 ) )
then
  Print( IntToStr( Bar ) );

```

15.18 PrintFlush

PrintFlush;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Causes any debug strings that have been issued via **Print** to be immediately displayed in the Debug window, **View|Debug Window (Ctrl+Alt+D)**. Normally, the debug strings are visible only after the ChartScript run completes.

Remarks

- Repetitive use of **PrintFlush** can increase a ChartScript's execution time significantly.

15.19 PrintStatus

PrintStatus(Value: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Causes the specified string to be immediately displayed in the lower-left status bar of Wealth-Lab Developer 4.0.

Remarks

- **View | Status Bar** must be selected from the main menu for the Status Bar to be visible (default).

Example

```
var SSYM: string;
var WL, BAR: integer;

for WL := 0 to WatchListCount -1 do
begin
  sSym := WatchListSymbol( WL );
  PrintStatus( 'Now Processing ' + sSym );
  SetPrimarySeries( sSym );
  for Bar := 20 to BarCount - 1 do
  begin
    { ... more statements ... }
  end;
end;
```

15.20 RunProgram

RunProgram(ProgramName: string; Wait: boolean);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Executes the specified program, *ProgramName*. If the *Wait* parameter is true, the script resumes executing only after the program terminates. Otherwise the script continues executing immediately after the program is launched.

Example

```
ShowMessage( 'About to launch Notepad ...' );
RunProgram( 'Notepad.exe', true );
ShowMessage( 'Notepad Closed!' );
```

15.21 SaveChartImage

SaveChartImage(FileName: string; Width: integer; Height: integer; ImageType: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Saves the current chart image to either a Bitmap or a GIF file. Specify the file name in the first parameter, *FileName*. Specify how large the resulting image should be by supplying values to the *Width* and *Height* parameters. The last parameter, *ImageType*, should contain either 'BMP' or 'GIF' which specifies which image format to save the chart.

Remarks

- **Available from the ChartScript window only.**
- Works for Real-Time ChartScript windows as well as for static data.

Saving Multiple Chart Images

The script below demonstrates how you can perform a batch job in WealthScript using the **WL3** COM Automation object **ExecuteScript** method. Open a new ChartScript Window (Ctrl+N) and then copy, and paste the code into the Editor. When you click on a symbol, the ChartScript specified in the *ScriptName* constant will be executed for each symbol in the WatchList. Consequently, by placing **SaveChartImage** in the *ScriptName* script ('Glitch Index' here), a chart image will be saved for each symbol. The name of the resulting file should be variable by symbol so that the same file is not continuously overwritten.

```
{ $NO_AUTO_EXECUTE }
const ScriptName = 'Glitch Index';
var obj: COMVariant;
var w: integer;
var sym, WatchList: string;

WatchList := WatchListName;
obj := CreateOleObject( 'WealthLab.WL3' );
for w := 0 to WatchListCount - 1 do
begin
    sym := WatchListSymbol( w );
    obj.ExecuteScript( ScriptName, WatchListName, sym );
end;
```

Example

```
{ Use 'as is' or paste at the bottom of any script to employ the batch
method above }
var str: string = GetSymbol + '_' + IntToStr( GetDate( BarCount - 1 ) )
+ '_' + GetScriptName;

{ Note! The folder specified here must exist }
SaveChartImage( 'C:\Data\Images\' + str + '.GIF', 600, 400, 'GIF' );
```

15.22 SetGlobal

SetGlobal(VariableName: string; Value: variant);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Allows you to store a variable, *VariableName*, and its *Value* in Wealth-Lab's global storage area. Use **GetGlobal** to retrieve the *Value* of the global variable assigned by **SetGlobal**.

Remarks

- Variables stored in the global storage area retain their values indefinitely, even between script calls.
- The name of the variable is specified in the *VariableName* parameter. If the variable already exists in the global storage area it will be overwritten.
- Specify the value in the *Value* parameter. Since the *Value* parameter is of type **Variant**, you can store values of any type (**integer**, **string**, **float** or **boolean**) in the global storage area.

Example

```
{ Store the BarCount by symbol }
var s: string;
s := GetSymbol + 'BarCount';
SetGlobal( s, BarCount );
```

15.23 SetOptimizeValue

SetOptimizeValue(OptValue : float);

ChartScripts SimuScripts PerfScripts CMScripts

Description

SetOptimizeValue accepts a single parameter, *OptValue*, which is used as the "User Defined" optimization metric. Typically, following the trading loop, you will cycle through the Positions to calculate some sort of trading metric and pass it as the *OptValue* parameter. This calculated value is added to the Optimization Results tabulations and graphs for both Exhaustive and Monte Carlo techniques, and, when running MC optimizations you may choose *User Defined* as the optimization target.

In the example we demonstrate a simple long-only, weighted-moving average crossover system with an 8% stop loss. The script is ready to optimize over the slow and fast periods of the moving average, and at the end, it calculates the average number of days in a trade for the User Defined metric.

Note: To see the optimization results, you must save this script and then Open the ChartScript for Optimization, **Ctrl+T**. Then from the Optimization view, select either Exhaustive or Monte Carlo methods. If Monte Carlo, you can further choose User Defined as the Optimization Metric, or target. Finally, click **Begin** to start the optimization process.

Remarks

- If a WatchList having more than one security is selected for a Portfolio \$imulation Mode optimization (*Portfolio \$imulation Mode* checked in the Optimization Control frame), the User-Defined value will be displayed as an average value.

Example

```
{#OptVar1 24;20;32;4}
{#OptVar2 14;10;18;2}
var Bar: integer;
var SlowPer, FastPer, SlowMA, FastMA: integer;
```

```

SlowPer := #OptVar1;
FastPer := #OptVar2;
SlowMA := WMAseries( #Close, SlowPer );
FastMA := WMAseries( #Close, FastPer );

InstallStopLoss( 8 );
PlotStops;
for Bar := SlowPer to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  if not LastPositionActive then
  begin
    if CrossOver( Bar, FastMA, SlowMA ) then
      BuyAtMarket( Bar + 1, 'CrossedOver' );
    end
  else
    if CrossUnder( Bar, FastMA, SlowMA ) then
      SellAtMarket( Bar + 1, LastPosition, 'CrossedUnder' );
    end;
  end;

  { Calculate Avg Days in Trade as User Defined optimization metric }
  var p, TotalDays: integer;
  for p := 0 to PositionCount - 1 do
    TotalDays := TotalDays + PositionExitBar( p ) - PositionEntryBar( p );
  end;

  if PositionCount = 0 then
    SetOptimizeValue( 0.0 );
  else
    SetOptimizeValue( TotalDays / PositionCount );
  end;
end;

```

15.24 SetPeakTroughMode

SetPeakTroughMode(Mode: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Specifies the way the Peak/Trough functions calculate a reversal amount. You can base Peak/Trough calculations on a percentage reversal, or a raw value "point" reversal. The *Mode* parameter can be one of the following constants:

- #AsPercent** - Peak/Trough reversals are expressed as percentages (default behavior)
- #AsPoint** - Peak/Trough reversals values are calculated based on raw value, i.e. points

For example, imagine you are calculating Peak/Troughs based on a reversal of 10. In percentage mode, the underlying Price Series has to move 10 percent higher than the lowest point in order for a Trough to be recorded. Likewise, prices must move 10 percent lower than a recent high for a Peak to be recorded. If you change the Peak/Trough mode to **#AsPoint** the underlying Price Series must move up or down 10 points rather than 10 percent.

Remarks

The **#AsPoint** mode is especially useful when calculating Peak/Troughs based on Price Series that can have zero and negative values. Percentage Peak/Trough reversals cannot be calculated on such Price Series.

Example

```

{ Draw Peaks/Troughs of an indicator that can enter the negative range
}
var P, T: float;
var REV, INDICATOR, PANE, PB_, PB, TB: integer;
Rev := 5;
Indicator := CMOSeries( #Close, 20 );
Pane := CreatePane( 150, true, true );
PlotSeries( Indicator, Pane, #Blue, #Thick );
PB_ := 0;
SetPeakTroughMode( #AsPoint );
PB := PeakBar( BarCount - 1, Indicator, Rev );
P := Peak( BarCount - 1, Indicator, Rev );
while ( PB <> PB_ ) and ( PB > 0 ) do
begin
  TB := TroughBar( PB, Indicator, Rev );
  T := Trough( PB, Indicator, Rev );
  DrawLine( PB, P, TB, T, Pane, #Red, #Thick );
  PB_ := PB;
  PB := PeakBar( TB, Indicator, Rev );
  P := Peak( TB, Indicator, Rev );
  DrawLine( PB, P, TB, T, Pane, #Red, #Thick );
end;

```

15.25 ShowMessage

```
ShowMessage( Message: string );
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Displays the specified *Message* in a dialog box. The ChartScript flow of execution is suspended until the user clicks on the dialog.

Example

```

if PositionProfit( LastPosition ) < 1000 then
begin
  ShowMessage( 'Time to Throw in the Towel!' );
  Abort;
end;

```

15.26 Sleep

```
Sleep( Milliseconds: integer );
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Causes the script to pause processing for the specified number of *Milliseconds*. You can abort a script that's sleeping by pressing the **Esc** key, or selecting **Chart|Stop Execution** from the main menu.

Note: The **Sleep** function is not compatible with ChartScript Integrated Debugger.

Example

```
Sleep( 2000 );
```

15.27 UseUpdatedEMA

UseUpdatedEMA(Use: boolean);

ChartScripts SimuScripts PerfScripts CMScripts

Description

This function controls which exponent formula is used when computing an **EMA** (Exponential Moving Average). By default, Wealth-Lab uses the following formula to calculate the exponent:

$$(1 / \text{Periods}) * 2$$

Another common method of calculating the Exponent is:

$$2 / (1 + \text{Periods})$$

Passing true as the *Use* parameter to **UseUpdatedEMA** will cause Wealth-Lab to use the second form when computing EMA's. This also affects native indicators that are based on **EMA**, such as **CADO**, **DSS**, **TRIX**, **MACD**, **Volatility**, etc.

Remarks

- EMA-based native indicators having the same parameters, e.g., `EMASeries(#Close, 100)` and `EMASeries(#Close, 100)`, will utilize the **UseUpdatedEMA** setting that is active at the time of the first reference to the indicator only. If you require both settings for the same EMA series, see the example for a solution.
- You can set the default preference for the EMA exponent calculation in the **Options Dialog (F12)|Indicator Calculations**. **UseUpdatedEMA** overrides the default setting.

Example

```
{ This script shows the difference between the 2 EMA exponent forms }
var EMASER, COPIED, EMASER2: integer;
UseUpdatedEMA( false );
EMASer := EMASeries( #Close, 12 );
Copied := AddSeriesValue( #Close, 0 );
UseUpdatedEMA( true );
EMASer2 := EMASeries( Copied, 12 );
PlotSeries( EMASer, 0, #Red, #Thin );
PlotSeries( EMASer2, 0, #Blue, #Thin );
```

15.28 WatchListAddSymbol

WatchListAddSymbol(Name: string; DSName: string; Symbol: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Adds the specified *Symbol* to the WatchList specified in the *Name* parameter. The *DSName* parameter is optional. If provided, it should contain the name of the DataSource that contains the Symbol being added. If left blank, the value defaults to the DataSource of the symbol that was clicked to execute the script.

Remarks

- **Available from the ChartScript window only.**
- The Symbol will not be added to the WatchList if it already exists - even if it was

from a different DataSource.

- **WatchListAddSymbol** creates the WatchList specified in the *Name* parameter if it does not already exist.

Example

```
{ Create a WatchList of the 5 most oversold symbols }
var w: integer;
var lst: TList;
var sym: string;
var x: float;

lst := TList.Create;
WatchListClear( 'Oversold' );

for w := 0 to WatchListCount - 1 do
begin
    sym := WatchListSymbol( w );
    SetPrimarySeries( sym );
    x := RSI( BarCount - 1, #Close, 20 );
    lst.AddData( x, sym );
end;

lst.SortNumeric;
for w := 0 to 4 do
begin
    sym := lst.Data( w );
    WatchListAddSymbol( 'Oversold', '', sym );
end;
```

15.29 WatchListClear

WatchListClear(Name: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Clears all of the symbols in the WatchList specified in the *Name* parameter.

Remarks

- **Available from the ChartScript window only.**
- The *Name* string is not case sensitive.

Example

```
{ Create a WatchList of all symbols where price is
above 200 day moving average }
var Bar, w: integer;
var sym: string;
WatchListClear( 'Above 200 Day SMA' );
for w := 0 to WatchListCount - 1 do
begin
    sym := WatchListSymbol( w );
    SetPrimarySeries( sym );
    Bar := BarCount - 1;
    if PriceClose( Bar ) > SMA( Bar, #Close, 200 ) then
        WatchListAddSymbol( 'Above 200 Day SMA', '', sym );
end;
```

15.30 WatchListCount

WatchListCount: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the number of symbols available in the currently selected WatchList.

Remarks

- Not compatible with real-time ChartScript mode.
- WatchList functions are generally intended for use in the ChartScript Window only.

Exceptions:

1. **WatchListCount** cannot be used with **SetPrimarySeries** in the \$imulator if you create trades before calling **RestorePrimarySeries**. This will cause the **\$imulator** to stop processing after the first symbol. A ChartScript that does not trade on secondary symbols can call **WatchListCount** to create an index indicator, for example.
2. If the ChartScript does create trades on secondary symbols, you can force **End-of-day Scans** to complete an entire scan by selecting "Multi-Symbol Script Scanning".

Example

```
{ Create an analysis file for all symbols in the WatchList.
  Output the RSI level and net gain after 20 bars }
var n, f, Bar: integer;
var val, change: float;
f := FileCreate( 'WatchList RSI Analysis.csv' );
for n := 0 to WatchListCount - 1 do
begin
  SetPrimarySeries( WatchListSymbol( n ) );
  Bar := 20;
  while Bar < BarCount - 20 do
  begin
    val := RSI( Bar, #Close, 20 );
    change := PriceClose( Bar + 20 ) - PriceClose( Bar );
    change := ( change / PriceClose( Bar ) ) * 100;
    FileWrite( f, GetSymbol + ',' + IntToStr( Bar ) + ',' +
      FloatToStr( val ) + ',' + FloatToStr( change ) );
    Bar := Bar + 20;
  end;
end;
```

15.31 WatchListDelete

WatchListDelete(Name: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Deletes the WatchList specified by the *Name* parameter completely. The *Name* string is not case sensitive.

Remarks

- After running a ChartScript that contains **WatchListDelete**, to see the result in the same ChartScript window you must refresh the DataSource tree by right clicking within it and selecting **Refresh**.
- If the specified WatchList does not exist, no action is taken and the statement is executed without error.
- **WatchListDelete** will not delete an actual *DataSource*. However, after passing a *DataSource Name*, it will *appear to be deleted* after refreshing the DataSource tree in the ChartScript window. This is because Wealth-Lab creates a mirrored WatchList for each DataSource. You can recover such WatchLists by simply restarting Wealth-Lab.

Example

```
{ Delete the WatchList created by the WatchListAddSymbol example }  
WatchListDelete( 'Oversold' );
```

15.32 WatchListName

WatchListName: string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns WatchList name containing the symbol that was clicked to execute the script.

Example

```
DrawLabel( 'The WatchList is: ' + WatchListName, 0 );
```

15.33 WatchListRemoveSymbol

WatchListRemoveSymbol(Name: string; Symbol: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Removes the specified *Symbol* from the WatchList specified by the *Name* parameter.

Remarks

- **Available from the ChartScript window only.**

15.34 WatchListSelect

WatchListSelect(WatchList: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Executes the ChartScript on the specified *WatchList*. If the ChartScript is currently being executed on a symbol in a WatchList different from the one specified in the *WatchList* parameter, this function aborts the script processing, selects the first symbol in the specified WatchList tree folder, and re-executes the script on this symbol. If the script is already being executed on a symbol within the selected *WatchList*, processing continues normally.

Remarks

- **Available from the ChartScript window only.**
- Not compatible with real-time ChartScript mode.

Example

```
{ Make sure the script executes in a specific Intraday WatchList }  
WatchListSelect( 'QCharts 15 Minute' );
```

15.35 WatchListSymbol

WatchListSymbol(n: integer): string;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the specified symbol from the currently selected WatchList. You can use this function, combined with **WatchListCount**, to write ChartScripts that act on all of the symbols in a WatchList.

Remarks

- Not compatible with real-time ChartScript mode.
- WatchList functions are generally intended for use in the ChartScript Window only.

Exceptions:

1. **WatchListSymbol** cannot be used with **SetPrimarySeries** in the \$imulator if you create trades before calling **RestorePrimarySeries**. This will cause the **\$imulator** to stop processing after the first symbol.
2. If the ChartScript does create trades on secondary symbols, you can force **End-of-day Scans** to complete an entire scan by selecting "Multi-Symbol Script Scanning".

Example

```
{ The following script executes for each symbol in the WatchList }  
var n: integer;  
for n := 0 to WatchListCount - 1 do  
begin  
  SetPrimarySeries( WatchListSymbol( n ) );  
  { ... }  
end;
```

16 Technical Indicator Functions

16.1 Overview

Technical Indicators make technical analysis possible. The indicators found in this reference are native to WealthScript, but by no means include all of the indicators available to Wealth-Lab users! You have the ability to create your own *Custom Indicators*, and many such indicators are uploaded by Wealth-Lab users every week. These will be downloaded directly to your Studies folder when you perform the **Community|Download ChartScripts** action.

To use either native or custom indicators in your ChartScripts, you can use the QuickPlot utility by dragging and dropping them right from the main Indicators tool bar. You can also use the Include Manager, **Tools|Include Manager (F6)**, to make a reference to a custom indicator yourself, although Quickplot adds these references automatically.

Indicators have two syntax forms. The first form returns the value of the indicator at a specific Bar number, and the second syntax form returns an integer *handle* reference to the indicator's entire Price Series. See Working with Technical Indicator Functions in the WealthScript Guide.

16.2 AccumDist

AccumDist(Bar: integer): float;
AccumDistSeries: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Accumulation/Distribution (created by L. Williams) uses the closing price's proximity to the high or low to determine if accumulation or distribution is occurring in the market. The proximity value is multiplied by volume to give more weight to moves with higher volume.

You can often spot divergences between price action and the AccumDist indicator. For example, if prices make a new high but the move is not accompanied by sufficient volume, AccumDist will fail to make a new high. Divergences can be a sign the trend is nearing completion.

Interpretation

The actual value of the AccumDist is unimportant, concentrate on its direction.

- When both price and AccumDist are making higher peaks and higher troughs, the up trend is likely to continue.
- When both price and AccumDist are making lower peaks and lower troughs, the down trend is likely to continue.
- When price continues to make higher peaks and AccumDist fails to make higher peak, the up trend is likely to stall or fail.
- When price continues to make lower troughs and AccumDist fails to make lower troughs, the down trend is likely to stall or fail.
- If during a trading range, the AccumDist is rising then accumulation may be taking place and is a warning of an upward break out.
- If during a trading range, the AccumDist is falling then distribution may be taking place and is a warning of an downward break out.

Calculation

$$\text{AccumDist} = (((\text{Close} - \text{Low}) - (\text{High} - \text{Close}) / (\text{High} - \text{Low})) \times \text{Volume}) + I$$

I = yesterday's AccumDist value

Example

```
{ Look for a diverging slope of AccumDist and price }
var BAR: integer;
for Bar := 10 to BarCount - 1 do
begin
  if AccumDist( Bar ) > AccumDist( Bar - 10 ) then
    if PriceClose( Bar ) < PriceClose( Bar - 10 ) then
      SetBarColor( Bar, #Red );
end;
var AccumDistPane: integer;
AccumDistPane := CreatePane( 100, false, true );
PlotSeries( AccumDistSeries, AccumDistPane, 202, #Thick );
DrawLabel( 'AccumDist', AccumDistPane );
```

16.3 ADX

ADX(Bar: integer; Period: integer): float;
 ADXSeries(Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

ADX stands for Average Directional movement Index and is used to measure the overall strength of the trend. The ADX indicator is an average of DX values, see DX. The ADX is a component of the Directional Movement System developed by Welles Wilder. This system attempts to measure the strength of price movement in positive and negative direction using the DIPlus and DIMinus indicators along with the ADX.

Interpretation

- The ADX is an excellent indicator for showing trend strength. The larger its value the stronger the current trend. A value above 25 is considered to be a trending market.

- When the ADX turns down from high values, then the trend maybe ending. It might be a good time to start closing open positions.
- If the ADX is declining, then the market is becoming less directional, the current trend is weakening. You should not be trading a trend system.
- When the ADX stays at a low value, the market is considered to be flat or dull. The longer the ADX stays at a low value the more likely a strong trending move will occur.
- If after staying low for a lengthy time, the ADX rises by 4 or 5 units, (for example, from 15 to 20), it gives a strong signal to trade the current trend.
- If the ADX is rising then the market is showing a strengthening trend. The value of the ADX is proportional to the slope of the trend. The slope of the ADX line is proportional to the acceleration of the price movement (changing trend slope). If the trend is a constant slope then the ADX value tends to flatten out.

Calculation

ADX is equivalent to the Wilder's moving average (see WilderMA) of the direction movement (DX) over the specified *Period*.

Example

```
{ Use ADX to determine how much prices are trending, color bars
accordingly }
var BAR, n: integer;
var x: float;
for Bar := 20 to BarCount - 1 do
begin
  x := ADX( Bar, 20 );
  n := Round( x / 5 );
  SetBarColor( Bar, n * 100 );
end;
```

16.4 ADXR

ADXR(Bar: integer; Period: integer): float;
ADXRSeries(Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

ADXR stands for Average Directional Movement Index Rating, and is a component of the Directional Movement System developed by Welles Wilder. This system attempts to measure the strength of price movement in positive and negative directions, as well as the overall strength of the trend. The ADXR component is simply a special type of moving average (WilderMA) applied to the ADX indicator.

The ADXR can be used to determine if price movement is sufficiently directional to be worth trading. In other words, use the ADXR as a filter to trade with trend following tools.

Interpretation

- ADXR is sometimes used as a signal line. A buy signal occurs when ADX crosses above ADXR, and a sell occurs when ADX crosses below ADXR.

- Welles Wilder's rule is to use trend follow systems when ADXR is above 25 and when ADXR drops below 20 then do not use a trend following system.
- ADXR behaves like an Averaged ADX. See ADX. The ADXR is a lagging indicator and will give signals after the ADX.
- The ADXR can be used in place of the ADX in the Directional Movement system. It results in more conservative trading signals.

Calculation

$$\text{ADXR} = (\text{ADX}(\text{today}) + \text{ADX}(\text{n days ago})) / 2$$

Example

```
{ Flag ADX/ADXR CrossOvers }
var BAR: integer;
for Bar := 20 to BarCount - 1 do
  if CrossOver( Bar, ADXSeries( 14 ), ADXRSeries( 14 ) ) then
    SetBackgroundColor( Bar, #RedBkg );
```

16.5 AroonDown

AroonDown(Bar: integer; Series: integer; Period: integer): float;
AroonDownSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Aroon indicator developed by Tushar Chande, indicates if a price is trending or in range trading. It can also reveal the beginning of a new trend, its strength and also allows you to anticipate changes from trading ranges to trends. AroonDown and the AroonUp indicators are used together and combined are called the Aroon indicator.

AroonUp measures how long it has been since prices have recorded a new high within the specified period. If the current price is higher than the user defined number of periods before it, then the AroonUp value is %100. In other words, it's a new high for that period. If a new low occurred during the period then AroonDown will be zero. Otherwise it returns a percent value indicating the time since the new high occurred.

AroonDown measures how long it has been since prices have recorded a new low within the specified period. If the current price is lower than the user defined number of periods before it, then the AroonDown value is %100. In other words, it's a new low for that period. If a new high occurred during the period then AroonDown will be zero. Otherwise it returns a percent value indicating the time since the new low occurred.

Another indicator, the Aroon Oscillator, can be constructed by subtracting AroonDown from AroonUp.

Interpretation

Weakness in the market is indicated when AroonDown remains between 0 and 30 for an extended period of time. If AroonDown and AroonUp follow similar movement patterns, this is a sign of consolidation. Finally, AroonDown crossing below AroonUp is considered a bearish sign.

- When AroonUp is at 100, a new uptrend may have begun. If it remains persistently between 70 and 100, and the AroonDown remain between 0 and 30, then a new uptrend is underway. If AroonUp dips below 50 then the trend as lost momentum.

- When AroonDown is at 100, a new downtrend may have begun. If it remains persistently between 70 and 100, and the AroonUp remain between 0 and 30, then a new downtrend is underway. If AroonDown dips below 50 then the trend as lost momentum.
- Trading ranges and consolidation. When AroonUp and AroonDown move in parallel (horizontal, sloping up or down) with each other at roughly the same level, then price is range trading or consolidating.
- New Trend, if the AroonUp crosses above the AroonDown, then a new uptrend may soon start. Conversely, if AroonDown crosses above the AroonUp, then a new downtrend may soon start.

Calculation

AroonUp:

$$100 * (n - (\text{Num. of bars since highest high in the last } n \text{ periods})) / n$$

AroonDown:

$$100 * (n - (\text{Num. of bars since lowest low in the last } n \text{ periods})) / n$$

n = number of periods or bars

Example

```
{ Flag Bearish Aroon Crossovers }
var Bar: integer;
for Bar := 20 to BarCount - 1 do
begin
  if CrossUnder( Bar, AroonDownSeries( #Close, 20 ),
                AroonUpSeries( #Close, 20 ) ) then
    SetBarColor( Bar, #Red );
end;
```

16.6 AroonUp

AroonUp(Bar: integer; Series: integer; Period: integer): float;
 AroonUpSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

See [AroonDown](#)^[17]

Example

```
{ Flag Bullish Aroon Crossovers }
var Bar: integer;
for Bar := 20 to BarCount - 1 do
begin
  if CrossOver( Bar, AroonDownSeries( #Close, 20 ),
               AroonUpSeries( #Close, 20 ) ) then
    SetBarColor( Bar, #Lime );
end;
```

16.7 ATR

ATR(Bar: integer; Period: integer): float;
 ATRSeries(Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Average True Range is the average of the true ranges over the specified Period. WealthScript uses the moving average as formulated by Welles Wilder, the indicator's inventor (see WilderMA and TrueRange). The ATR is a measure of volatility and it takes into account any gaps in the price movement. Typically the ATR calculation is based on 14 periods, this can be intraday, daily, weekly or monthly. To measure recent volatility use a shorter average, 2 to 10 periods. For longer term volatility use 20 to 50 periods.

Interpretation

- An expanding ATR indicates increased volatility in the market. The range of each bar is getting larger. ATR often peaks at major tops and bottoms. High ATR values usually result from a sharp advance or decline and are unlikely to be sustained for extended periods.
- A low average true range value indicates a series of periods with small ranges (quiet days). These low ATR values are often found during extended sideways price action, thus lower volatility. A prolonged period of low ATR values may indicate a consolidation area and the beginning of a continuation move or reversal.
- ATR is very useful for stops or entry triggers, as it allows for changes in volatility. Whereas fixed dollar, points or percentage stops will not allow for volatility. The ATR stop will adapt to sharp price moves or consolidation areas, and trigger on an abnormal price movement in either area. Use a multiple of ATR, such as 1.5 x ATR(5 period) to catch these abnormal price moves.

Calculation

Average True Range is calculated by applying Wilder's Moving Average to True Range over the period specified, see WilderMA indicator for more information:

$$\text{ATR} = (\text{Previous ATR} * (n - 1) + \text{TR}) / n$$

where,

ATR = Average True Range
 n = number of periods or bars
 TR = True Range, (see **TrueRange** indicator)

Example

```
{ Plot ATRs in decreasing length if increasing blue intensity }
var i, ATRPane: integer;
ATRPane := CreatePane( 100, TRUE, TRUE );
for i := 1 to 9 do
  PlotSeries( ATRSeries( i * 2 ), ATRPane, 10 - i, #Thin );
DrawText( 'ATR from 2 to 18', ATRPane, 4, 4, 006, 8 );
```


16.8 ATRP

ATRP(Bar: integer; Period: integer): float;
 ATRPSeries(Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

ATRP expresses the Average True Range, or ATR, as a percentage of the closing price of the specified *Bar*. **ATRP** provides a good picture of current volatility.

Calculation

$$\text{ATRP}(\text{Bar}, \text{Period}) = 100 * \text{ATR}(\text{Bar}, \text{Period}) / \text{PriceClose}(\text{Bar})$$

where,

ATR = Average True Range (see indicator)

Example

```
{ Short when price hits the High of the previous bar * (1 + ATRP/100 )
  Cover on trailing stop of the same series }
var Bar, hATRP, hATRP_H, p: integer;

{ Convert to fractional percentage, e.g., 3.5% -> 0.035 }
hATRP := DivideSeriesValue( ATRPSeries( 5 ), 100 );
hATRP := AddSeriesValue( hATRP, 1.0 );
hATRP_H := MultiplySeries( #High, hATRP );

{ Delay indicator plot by 1 bar to observe crossovers }
PlotSeriesLabel( OffsetSeries( hATRP_H, -1), 0, #Blue, #Dotted, 'ATRP_H
+ 2%' );
PlotStops;

for Bar := 5 to BarCount - 1 do
begin
  if LastPositionActive then
  begin
    p := LastPosition;
    CoverAtTrailingStop( Bar + 1, @hATRP_H[Bar], p, '' )
  end
  else
  ShortAtLimit( Bar + 1, @hATRP_H[Bar], '' );
end;
```

16.9 BBandLower

BBandLower(Bar: integer; Series: integer; Period: integer; StdDev: float): float;
 BBandLowerSeries(Series: integer; Period: integer; StdDev: float): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Bollinger Bands are a type of price envelope developed by John Bollinger. Bollinger Bands are envelopes that are plotted at a standard deviation level above and below a simple moving average of the price. Because the distance of the bands is based on standard deviation, they adjust to volatility swings in the underlying price.

Bollinger Bands accept 2 parameters, *Period* and Standard Deviations, *StdDev*. The recommended values are 20 for period, and 2 for standard deviations, although other combinations offer effective results as well.

Bollinger bands help determine whether prices are high or low on a relative basis. They are used in pairs, both upper and lower bands and in conjunction with a moving average. Further, the pair of bands are not intended to be used on their own. Use them to confirm signals given with other indicators. For example, RSI and Bollinger bands are a good combination.

Interpretation

- When the bands tighten as volatility decreases, expect a sharp move in price. This may begin a trending move. Watch out for a false move in opposite direction which reverses before the proper trend begins.
- When the bands separated by an unusual large amount, volatility increases and any trend that may be in place may be ending.
- Prices normally have a tendency to bounce within the bands envelope, touching one band then moving to the other band. You can use this for profit targets. For example, if prices bounces of the lower band then cross above the moving average the upper band then becomes the profit target.
- Price can exceed or hug a band envelope for prolonged periods during strong trends. On divergence with a momentum oscillator you should consider taking profits.
- A strong trend continuation can be expected when the price moves out of the bands. However if prices move immediately back inside the band, then the suggested strength is negated.

Calculation

First calculate and plot a simple moving average. Calculate the standard deviation using the same data used in the simple moving average. For the upper band, add the standard deviation to the moving average, for lower band, subtract the standard deviation from the moving average.

Typical values used:

Short term: 10 day moving average, bands at 1.5 standard deviations.
 Medium term: 20 day moving average, bands at 2 standard deviations.
 Long term: 50 day moving average, bands at 2.5 standard deviations.

Example

```
{ Flag bars that have penetrated the lower BBand }
var Bar: integer;
PlotSeries( BBandLowerSeries( #Close, 10, 1.50 ), 0, 205, #Thick );
for Bar := 10 to BarCount - 1 do
  if PriceLow( Bar ) < BBandLower( Bar, #Close, 10, 1.50 ) then
    SetBarColor( Bar, #Red );
```

16.10 BBandUpper

BBandUpper(Bar: integer; Series: integer; Period: integer; StdDev: float): float;
 BBandUpperSeries(Series: integer; Period: integer; StdDev: float): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

See [BBandLower](#)^[174]

Example

```
{ Flag bars that have penetrated the upper BBand }
var Bar: integer;
PlotSeries( BBandUpperSeries( #Close, 10, 1.50 ), 0, 205, #Thick );
for Bar := 10 to BarCount - 1 do
  if PriceHigh( Bar ) > BBandUpper( Bar, #Close, 10, 1.50 ) then
    SetBarColor( Bar, 050 );
```

16.11 BOP

BOP(Bar: integer): float;
BOPSeries: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Balance of Power, created by Igor Livshin, is an indicator that captures the struggles of bulls vs. bears throughout the trading day. It assigns scores to both bulls and bears based on how much they were able to move prices throughout the trading day.

Interpretation

- Balance of Power is normally smoothed with a moving average. Livshin recommends a 14 bar simple moving average, but different periods and moving average types can be used for different markets.
- During Bull markets, the indicator's tops usually cluster around the upper level of the range. This is reversed during Bear markets.
- You can look for divergences between the indicator and the underlying price to spot potential trend reversals.

Calculation

Balance of Power is the result of the following simple formula:

$$\text{BOP} = (C - O) / (H - L)$$

where,

C = Close, O = Open, H = High and L = Low

Example

```
{ Plot a smoothed Balance of Power below Volume }
var BOPSmoothed, BOPPane: integer;
BOPSmoothed := SMASeries( BOPSeries, 20 );
BOPPane := CreatePane( 80, false, true );
PlotSeries( BOPSmoothed, BOPPane, 642, #Thick );
```

16.12 CADO

CADO(Bar: integer): float;
CADOSeries: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Chaikin Oscillator allows us to analyze accumulation and distribution in the convenient form of an oscillator. The principle behind this oscillator is the nearer the close is to the high, the more accumulation taken place. Conversely the nearer the close is to the low, the more distribution is taken place. Further, healthy rallies and declines are accompanied by increasing volume levels, conversely price tends to decline as volume dries up. The Chaikin Oscillator allows you to compare price action to volume flow, to help determine market tops and bottoms.

Interpretation

- The best Chaikin Oscillator sell signal is when price action develops a higher high into overbought zones and the Chaikin Oscillator diverges with a lower high and begins to fall. Price may remain in overbought zones during strong trends.
- The best Chaikin Oscillator buy signal is when price action develops a lower low into oversold zones and the Chaikin Oscillator diverges with a higher low and begins to rise. Price may remain in oversold zones during strong trends.
- You can also use the Chaikin Oscillator to assist entry into existing trends. In this case you look for a change of direction of the oscillator for buy or sell signal. For example, if you have confirmed strong uptrend and the Chaikin Oscillator turns up from a negative value, then buy the dip in price action.

Calculation

The Chaikin Oscillator is created by subtracting a 10-period EMA of Accumulation/Distribution from a 3-period EMA of Accumulation/Distribution.

$$CADO = 3 \text{ period EMA}(\text{AccumDist}()) - 10 \text{ period EMA}(\text{AccumDist}())$$

where,

CADO = Chaikin Oscillator

EMA = Exponential Moving Average

AccumDist = Accumulation/Distribution indicator

Note: UseUpdatedEMA affects the calculation of EMA-based indicators such as **CADO** at runtime. Choose the default method for calculating the EMA exponent in the *Indicator Calculations* section of the Options dialog.

Example

```
{ Use RSI to look for divergences between price and CADO }
var Bar, Diff, CADOPANE, RSIPANE, RSIPRICE, RSICADO, DiffPane: integer;

CADOPane := CreatePane( 80, true, true );
PlotSeries( CADOSeries, CADOPane, 520, #ThickHist );
DrawLabel( 'CADO', CADOPane );

RSIPane := CreatePane( 100, true, true );
RSIPrice := RSISeries( #Close, 10 );
RSICado := RSISeries( CADOSeries, 10 );
PlotSeries( RSIPrice, RSIPane, #Teal, #Thin );
PlotSeries( RSICado, RSIPane, 520, #Thin );
DrawLabel( 'RSI of Price and CADO', RSIPane );

Diff := SubtractSeries( RSIPrice, RSICado );
DiffPane := CreatePane( 100, true, true );
PlotSeries( Diff, DiffPane, #Gray, #Histogram );

for Bar := 10 to BarCount - 1 do
begin
```

```

if GetSeriesValue( Bar, Diff ) > 30 then
  SetBarColor( Bar, #Red )
else if GetSeriesValue( Bar, Diff ) < -30 then
  SetBarColor( Bar, 050 );
end;

```

16.13 CCI

CCI(Bar: integer; Period: integer): float;
CCISeries(Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Commodity Channel Index (CCI) developed by Lambert, is designed to identify and trade cyclical turns in commodities. It assumes the commodity or stock moves in cycles. Lambert recommends using 1/3 of the cycle as the calculation period. The cycle is considered an interval of low-to-low or high-to-high. Commodities can cycle around 60 days, thus the period would be 20 days. Signals are given when CCI moves into the +100 or -100 regions.

Interpretation

- When the CCI moves above +100, then a new strong uptrend is beginning, buy here, close the position on CCI falling below +100. Use trending indicators or other technical analysis methods to confirm.
- When the CCI moves below -100, then a new strong downtrend is beginning, sell here, close the position on CCI rising above -100. Use trending indicators or other technical analysis methods to confirm.
- If underlying prices make a new high or low that isn't confirmed by the CCI, this divergence can signal a price reversal. CSI divergences from price indicates very strong buy or sell signal.
- Look for oversold levels below -100 and overbought levels above +100. These normally occur before the underlying price chart forms a top or a bottom.

Calculation

The Commodity Channel Index (CCI) is calculated by determining the difference between the mean price of a security and the average of the means over the period chosen. This difference is compared to the average difference over the time period. Comparing the differences of the averages allows for the commodities volatility. The result is multiplied by a constant to ensure that most values fall within the standard range of +/- 100.

$$CCI = (AveP - SMA_of_AveP) / (0.015 * Mean\ Deviation)$$

where,

$$CCI = \text{Commodity Channel Index}$$

$$AveP = \text{Average Price} = (High + Low + Close) / 3$$

The 0.015 constant ensures 70 to 80 percent of CCI values fall within the +100 to -100 range.

Example

```

{ Color bars oversold/overbought based on CCI level }
var Bar, CCIPane: integer;

```

```

CCIPane := CreatePane( 80, true, true );
PlotSeries( CCISeries( 10 ), CCIPane, 505, #Histogram );
DrawLabel( 'CCI(10)', CCIPane );
for Bar := 10 to BarCount - 1 do
begin
  if CCI( Bar, 10 ) > 100 then
    SetBarColor( Bar, #Red )
  else if CCI( Bar, 10 ) < -100 then
    SetBarColor( Bar, #Green );
end;

```

16.14 CMF

CMF(Bar, Period: integer): float;
 CMFSeries(Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Chaikin Money Flow (CMF) is a volume weighted average of Accumulation/Distribution over the specified period. The standard CMF period is 21 days. The principle behind the Chaikin Money Flow, is the nearer the close is to the high, the more accumulation has taken place. Conversely the nearer the close is to the low, the more distribution has taken place. If the price action consistently closes above the bar's midpoint on increasing volume then the Chaikin Money Flow will be positive. Conversely, if the price action consistently closes below the bar's midpoint on increasing volume, then the Chaikin Money Flow will be a negative value.

Interpretation

- A CMF sell signal occurs when price action develops a higher high into overbought zones and the CMF diverges with a lower high and begins to fall.
- A CMF buy signal occurs when price action develops a lower low into oversold zones and the CMF diverges with a higher low and begins to rise.
- A CMF value above the zero line is a sign of strength in the market, and a value below the zero line is a sign of weakness in the market.
- The Chaikin Money Flow provides excellent breakout confirmation. Wait for the CMF to confirm the breakout direction of price action through trendlines or support and resistance lines. For example, if price breaks upwards through resistance then wait for the CMF to have a positive value, thus confirming the break out direction.

Calculation

$$\text{CMF} = \frac{\text{n-day Sum of } (((C - L) - (H - C)) / (H - L)) \times \text{Vol})}{\text{n-day Sum of Vol}}$$

where,

n = number of periods, typically 21
 H = high
 L = low
 C = close
 Vol = volume

Example

```

{ Use strength of CMF above zero to color bars }
var Bar, CMFPane: integer;

```

```

var x: float;
CMFPane := CreatePane( 80, true, true );
PlotSeries( CMFSeries( 20 ), CMFPane, 509, #Histogram );
DrawLabel( 'CMF( 20 )', CMFPane );
for Bar := 20 to BarCount - 1 do
begin
  x := CMF( Bar, 20 ) * 20;
  if x > 0 then
    SetBarColor( Bar, Round( x ) );
end;

```

16.15 CMO

CMO(Bar: integer; Series: integer; Period: integer): float;
 CMOSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Chande Momentum Oscillator is similar to RSI or Stochastics. It is calculated by dividing the sum of up day and down day activity into the difference of up day and down day activity. The result is then multiplied by 100 to arrive at an indicator that oscillates between -100 and 100. A typical value for number of periods, *Period*, for the CMO is 20.

Interpretation

- CMO reaches extreme levels at 50 for overbought and -50 for oversold. You can also look for signals based on the CMO crossing above and below a signal line composed of a 9 period moving average of the 20 period CMO.
- CMO measures the trend strength, the higher the CMO value the stronger the trend, whereas low CMO values indicate sideways trading ranges.
- If underlying prices make a new high or low that isn't confirmed by the CMO this divergence can signal a price reversal.
- CMO often forms chart patterns which may not show on the underlying price chart, such as double tops and bottoms and trendlines. Also look for support or resistance on the CMO.

Calculation

$$\text{CMO} = 100 * ((\text{Su} - \text{Sd}) / (\text{Su} + \text{Sd}))$$

where,

Su = Sum of prices on up days for the specified *Period*
 Sd = Sum of prices on down days for the specified *Period*

Example

```

{ This simple system buys when CMO is oversold,
  and sells when CMO is overbought }
var Bar: integer;
for Bar := 20 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    if CMO( Bar, #Close, 20 ) < -55 then
      BuyAtMarket( Bar + 1, 'CMO' );
  end
end

```

```

else
begin
  if CMO( Bar, #Close, 20 ) > 45 then
    SellAtMarket( Bar + 1, LastPosition, 'CMO' );
  end;
end;

```

16.16 CumDown

CumDown(Bar: integer; Series: integer; Period: integer): float;
 CumDownSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

CumDown lets you test whether a specific number of consecutive bars have elapsed where the prices are less than their value a certain number of bars ago. It was created to make it easier to implement systems such as TD Sequential (by Thomas Demark). The TD Sequential setup requires 9 consecutive bars where the closing price is lower than the closing price 4 bars ago.

The complete TD system encompasses both entry and exit strategies and an extensive number of TD indicators. The CumUp and CumDown indicators are used to find setup conditions indicating overbought and oversold market conditions. They are designed to anticipate trend reversals. The CumUp looks for a number of new high periods with only a few low periods. The CumDown looks for a number of new low periods with only a few high periods.

In Candles sticks a new high or low is called Record Sessions. Candle theory suggests that if you have 8 to 10 near record sessions then the preceding trend is due for a reversal. Record sessions count the bars slightly differently than CumDown and CumUp.

Interpretation

There are three stages to a TD Sequential system, the Setup, the Intersection, and the Count down. After each stage is triggered, move onto the next stage. The following is for oversold markets.

- The buy Setup consists of a series of at least nine consecutive closes less than the close four trading bars earlier. This indicates a possible oversold market.
- The buy Intersection, look for the high of bar 8 of the buy setup to be greater than or equal to the low of bars 5, 4, 3, 2 or 1 of the buy setup. If this is not fulfilled, then each successive price bar is compared until its high is greater than or equal to the low of the price bar three or more price bars earlier back to bar 1 of the buy setup. Protects against runaway price action.
- The buy Countdown consists of a series of 13 successive closes less than or equal to the low two price bars earlier. Once that has been accomplished, the market generally is in a low-risk buy entry zone. Good time to go long.

In a similar manner, use CumUp to detect overbought conditions.

Calculation

CumDown is a running count of the number of bars whose *Series* value is below its delayed *Series*; in other words, *Series* offset forward by the *Period*. The count is reset to zero when the *Series* is above its offset series. Run the following ChartScript for a visual clarification:


```

{ CumDown calculation demo }
const L = 5;
var Bar, n: integer;

DrawLabel( 'CumDown(#Low,' + IntToStr( L ) + ')', 0 );
PlotSeriesLabel( OffsetSeries( #Low, -L ), 0, #Blue, #Dotted,
                 'Offset(#Low, -' + IntToStr( L ) + ')');
for Bar := L to BarCount - 1 do
begin
  n := Trunc( CumDown( Bar, #Low, L ) );
  AnnotateBar( IntToStr( n ), Bar, false, 0, 8 );
end;

```

Notice that as long as the *Series*, #Low, is below its CumDown-period offset series, **CumDown** is incremented. It is reset to 0 as soon as the *Series* rises above its offset.

Example

```

{ Highlight extreme moves down }
var Bar, n: integer;
for Bar := 0 to BarCount - 1 do
begin
  n := Trunc( CumDown( Bar, #Close, 3 ) );
  if n > 9 then
    n := 9;
  SetBarColor( Bar, n * 100 );
end;

```

16.17 CumUp

CumUp(Bar: integer; Series: integer; Period: integer): float;
 CumUpSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

CumUp lets you test whether a specific number of consecutive bars have elapsed where the prices are greater than their value a certain number of bars ago. Refer to [CumDown](#)^[187] for additional information.

Calculation

CumUp is a running count of the number of bars whose *Series* value is above its delayed *Series*; in other words, *Series* offset forward by the *Period*. The count is reset to zero when the *Series* is below its offset series. Run the following ChartScript for a visual clarification:

```

{ CumUp calculation demo }
const L = 5;
var Bar, n: integer;

DrawLabel( 'CumUp(#High,' + IntToStr( L ) + ')', 0 );
PlotSeriesLabel( OffsetSeries( #High, -L ), 0, #Blue, #Dotted,
                 'Offset(#High, -' + IntToStr( L ) + ')');
for Bar := L to BarCount - 1 do
begin
  n := Trunc( CumUp( Bar, #High, L ) );
  AnnotateBar( IntToStr( n ), Bar, true, 0, 8 );
end;

```

Notice that as long as the *Series*, #High, is above its CumUp-period offset series,

CumUp is incremented. It is reset to 0 as soon as the *Series* falls below its offset.

Example

```
{ Highlight extreme moves up }
var Bar, n: integer;
for Bar := 0 to BarCount - 1 do
begin
  n := Trunc( CumUp( Bar, #Close, 3 ) );
  if n > 9 then
    n := 9;
  SetBarColor( Bar, n * 10 );
end;
```

16.18 DIMinus

DIMinus(Bar: integer; Period: integer): float;
DIMinusSeries(Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

DIMinus is a component of the Directional Movement System developed by Welles Wilder. This system attempts to measure the strength of price movement in positive and negative directions, as well as the overall strength of the trend. DIPlus is normally used with the DIMinus, DX and ADX indicators and typically uses 14 periods.

The DIMinus value represents downward price movement as a percentage of true range. The more each down bar's price is equal to the true range, the larger the value of the DIMinus. The DIPlus and the DIMinus are not mirror images.

Interpretation

- DIMinus measures a market's negative directional movement. If DIMinus is greater than DIPlus, prices have a stronger negative directional movement.
- If prices fall for the number of periods specified then the DIMinus would be a high value and the DIPlus value would be near zero.
- If prices rise for the number of periods specified then the DIMinus value would be near zero and DIPlus would have a high value.
- If prices fluctuate for the number of periods specified, like in a trading range, then DIPlus and DIMinus will have similar values.
- The greater the difference between the DIPlus and DIMinus the stronger the trend. The DX indicator takes advantage of this.

Calculation

$$-DI = \text{Round}((-DM / TR) * 100)$$

where,

$$\begin{aligned} -DI &= \text{DIMinus} \\ TR &= \text{True Range of current bar} \end{aligned}$$

The -DI is then smoothed over the *Period* specified, the same way as a simple moving average, and, -DM is calculated as follows:

- For up trending days, -DM = zero
- For down trending days, -DM = yesterday's low - today's low

- (iii) For inside days, -DM = zero
- (iv) For outside days, if yesterday's low - today's low, is greater than today's high - yesterday's high, then -MD = yesterday's low - today's low, otherwise -DM = zero
- (v) For upwards gap days, -DM = zero
- (vi) For downwards gap days, -DM = yesterday's low - today's low

Example

```

{ Color bars green when DI+ > DI-, otherwise color them red }
var Bar: integer;
var ADXPane: integer;
ADXPane := CreatePane( 100, true, true );
PlotSeries( DIMinusSeries( 14 ), ADXPane, 900, #Thick );
DrawLabel( 'DIMinus( 14 )', ADXPane );
PlotSeries( DIPlusSeries( 14 ), ADXPane, 050, #Thick );
DrawLabel( 'DIPlus( 14 )', ADXPane );
for Bar := 14 to BarCount - 1 do
begin
  if DIPlus( Bar, 14 ) > DIMinus( Bar, 14 ) then
    SetBarColor( Bar, #Green )
  else
    SetBarColor( Bar, #Red );
end;

```

16.19 DIPlus

DIPlus(Bar: integer; Period: integer): float;
 DIPlusSeries(Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

DIPlus is a component of the Directional Movement System developed by Welles Wilder. This system attempts to measure the strength of price movement in positive and negative directions, as well as the overall strength of the trend. DIPlus is normally used with the DIMinus, DX and ADX indicators and typically uses 14 periods.

The DIPlus value represents upward price movement as a percentage of true range. The more each up bar's price is equal to the true range, the larger the value of the DIPlus. The DIPlus and the DIMinus are not mirror images.

Interpretation

- DIPlus measures a market's positive directional movement. If DIPlus is greater than DIMinus, prices have a stronger positive directional movement.
- If prices rise for the number of periods specified then the DIPlus would be a high value and the DIMinus value would be near zero.
- If prices fall for the number of periods specified then the DIPlus value would be near zero and DIMinus would have a high value.
- If prices fluctuate for the number of periods specified, like in a trading range, then DIPlus and DIMinus will have similar values.
- The greater the difference between the DIPlus and DIMinus the stronger the trend. The DX indicator takes advantage of this.

Calculation

$$+DI = \text{Round} \left(\left(\frac{+DM}{TR} \right) * 100 \right)$$

where,

DI+ = DIPlus

TR = True Range of current bar

The +DI is then smoothed over the period specified, the same way as a simple moving average, and +DM is calculated as follows:

- (i) For up trending days, +DM = today's high - yesterday's high
- (ii) For down trending days, +DM = zero
- (iii) For inside days, +DM = zero
- (iv) For outside days, if today's high - yesterday's high, is greater than yesterday's low - today's low, then +DM = today's high - yesterday's high, otherwise +DM = zero
- (v) For upwards gap days, +DM = today's high - yesterday's high
- (vi) For downwards gap days, +DM = zero

Example

```
{ Flag bars with dotted lines when DI+ is above 40 }
var Bar: integer;
var ADXPane: integer;
ADXPane := CreatePane( 100, true, true );
PlotSeries( DIPlusSeries( 14 ), ADXPane, 050, #Thick );
DrawLabel( 'DIPlus( 14 )', ADXPane );
for Bar := 14 to BarCount - 1 do
begin
  if DIPlus( Bar, 14 ) > 40 then
    DrawLine( Bar, PriceLow( Bar ), Bar, 0, 0, #Green, #Dotted );
end;
```

16.20 DSS

DSS(Bar: integer; Period1: integer; Period2: integer; StochPeriod: integer): float;
 DSSSeries(Period1: integer; Period2: integer; StochPeriod: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Double Smoothed Stochastic indicator was created by William Blau. It applies Exponential Moving Averages (EMAs) of two different periods to a standard Stochastic % K (StochK). The components that construct the Stochastic Oscillator are first smoothed with the two EMAs. Then, the smoothed components are plugged into the standard Stochastic formula to calculate the indicator.

Interpretation

DSS ranges from 0 to 100, like the standard Stochastic Oscillator. The same rules of interpretation that you use for Stochastics can be applied to DSS, although DSS offers a much smoother curve than the raw Stochastic.

Calculation

```
HH = Highest High in Look back Period
LL = Lowest Low in Look back Period
C-L = Close minus LL
H-L = HH minus LL
C-L(2) = EMA( C-L, Period2 )
H-L(2) = EMA( H-L, Period2 )
```

```
C-L(1) = EMA( C-L(2), Period1 )
H-L(1) = EMA( H-L(2), Period1 )
```

```
DSS = ( C-L(1) / H-L(1) ) * 100
```

Note: **UseUpdatedEMA** affects the calculation of EMA-based indicators such as **DSS**.

Example

```
{ Buy when DSS turns up from an oversold level }
var Bar, DSSPane: integer;
DSSPane := CreatePane( 100, true, true );
PlotSeries( DSSSeries( 10, 20, 5 ), DSSPane, 905, #Thick );
DrawLabel( 'DSS( 10, 20, 5 )', DSSPane );
for Bar := 20 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    if TurnUp( Bar, DSSSeries( 10, 20, 5 ) ) then
      if DSS( Bar - 1, 10, 20, 5 ) < 24 then
        BuyAtMarket( Bar + 1, 'DSS' );
    end
  else
  begin
    if TurnDown( Bar, DSSSeries( 10, 20, 5 ) ) then
      SellAtMarket( Bar + 1, LastPosition, 'DSS' );
    end;
  end;
end;
```

16.21 DX

```
DX( Bar: integer; Period: integer ): float;
DXSeries( Period: integer ): integer;
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

DX is a component of the Directional Movement System developed by Welles Wilder. This system attempts to measure the strength of price movement in positive and negative directions, as well as the overall strength of the trend. The DX is used in calculating the ADX indicator which is normally used with the DIPlus and DIMinus indicators. Typically it uses 14 periods in its calculations. Normally you would not use the DX indicator as it whipsaws, use the ADX or ADXR.

Interpretation

- DX measures the trendiness of a market, and ranges from 0 to 100. If the Trend is strong then the spread between the two smoothed directional lines, (DIPlus and DIMinus) increases and the DX value increases. The higher the DX, the more directional movement present in the market.
- If prices rise for the number of periods specified then the DIMinus value would be near zero and DIPlus would have a high value. This very directional upwards price movement results in a high DX value.
- If prices fall for the number of periods specified then the DIMinus would be a high value and the DIPlus value would be near zero. This very directional downwards price movement result in a high DX value.
- If prices fluctuate for the number of periods specified, like in a trading range, then DIPlus and DIMinus will have similar values. This non-directional sideways price movements results in a low DX value.

Calculation

$$DX = \text{Round}(100 * |DIPlus - DIMinus| / |DIPlus + DIMinus|)$$

Example

```
{ Show how Average Directional Movement (ADX) relates to DX on the
chart }
var ADXPane: integer;
ADXPane := CreatePane( 100, true, true );
PlotSeries( DXSeries( 20 ), ADXPane, 555, #ThickHist );
DrawLabel( 'DX( 20 )', ADXPane );
PlotSeries( ADXSeries( 20 ), ADXPane, 009, #Thick );
DrawLabel( 'ADX( 20 )', ADXPane );
```

16.22 EMA

EMA(Bar: integer; Series: integer; Period: integer): float;
 EMASeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

EMA returns the Exponential Moving Average of the specified *Period*. EMA is similar to Simple Moving Average (SMA), in that it averages the data over a period of time. However, whereas SMA just calculates a straight average of the data, EMA applies more weight to the data that is more current. The most weight is placed on the most recent data point. Because of the way it's calculated, EMA will follow prices more closely than a corresponding SMA.

Note: UseUpdatedEMA affects the calculation of EMA-based indicators such as **CADO, EMA, TRIX**, etc.

Interpretation

- Use the same rules that we apply to SMA when interpreting EMA. Keep in mind, though, that EMA is generally more sensitive to price movement. This can be a double-edged sword. On the one hand, it can get you into trends a bit earlier than an SMA would. On the other hand, the EMA will probably experience more whipsaws than a corresponding SMA.
- Use the EMA to determine trend direction, and trade in that direction. When the EMA rises then buy when prices dip near or a bit below the EMA. When the EMA falls then sell when prices rally towards or a bit above the EMA.
- Moving averages can also indicate support and resistance areas. A rising EMA tends to support the price action and a falling EMA tends to provide resistance to price action. This reinforces the idea of buying when price is near the rising EMA or selling when price is near the falling EMA.
- All Moving Averages, including the EMA are not designed to get you into a trade at the exact bottom and out again at the exact top. They tend to ensure your trading in the general direction of the trend, but with a delay at the entry and exit. The EMA has a shorter delay than the SMA with the same period.

Calculation

You should notice how the EMA use the previous value of the EMA in its calculation, this means the EMA includes all the price data within its current value. The newest price data has the most impact on the Moving Average and the oldest prices data has only a minimal impact.

$$\text{EMA} = (K \times (C - P)) + P$$

where,

C = Current Price

P = Previous periods EMA (A SMA is used for the first periods calculations)

K = Exponential smoothing constant

The smoothing constant K, applies appropriate weight to the most recent price. It uses the number of periods specified in the moving average. With wealth Lab you have a choice of two methods for calculating the smoothing constant.

Two similar *but not equivalent* formulas are available for calculating the exponent; Wealth-Lab's original method (from Pring's *Technical Analysis Explained*):

$$K = (1 / \text{Periods}) * 2$$

and perhaps a more common method, which is referred to as the "Updated Method":

$$K = 2 / (1 + \text{Periods})$$

You can choose which is the default method in the *Indicator Calculations* section of the Options dialog. Additionally, by passing true or false to the **UseUpdatedEMA** function you can control which method is used at runtime. Note that the formula in effect also affects native indicators that are based on **EMA**, such as **CADO**, **TRIX**, etc.

Example

```
{ Dual EMA CrossOver System }
var BAR, P: integer;
{ UseUpdatedEMA( true ); } {Alternate smoothing exponent}
PlotSeries( EMASeries( #Close, 60 ), 0, 002, #Thick );
DrawLabel( 'EMA( Close, 60 )', 0 );
PlotSeries( EMASeries( #Close, 120 ), 0, 202, #Thin );
DrawLabel( 'EMA( Close, 120 )', 0 );
for Bar := 120 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    if CrossOver( Bar, EMASeries( #Close, 60 ), EMASeries( #Close, 120
) ) then
      BuyAtMarket( Bar + 1, '' );
    end
  else
  begin
    if CrossUnder( Bar, EMASeries( #Close, 60 ), EMASeries( #Close, 120
) ) then
      SellAtMarket( Bar + 1, LastPosition, '' );
    end;
  end;
end;
```

16.23 EMMinus

EMMinus(Bar: integer; Series: integer; Period: integer): float;
EMMinusSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Markets that are experiencing rising trends frequently make new highs, and those in falling trends new lows. The Extreme Motion Index is a way of measuring a market's trend strength by counting the frequency of new highs and lows in a given period. EM- is the percentage of bars that have made new lows within the specified *Period*.

Interpretation

- When EM- rises from zero this is an indication that a worthwhile downtrend may be in the making. You can take a trend-following position at this point and exit once the indicator reaches a predetermined overbought level.
- Once EM- crosses 20 prices tend to follow through and a profit target or trailing stop (for short orders) often works well to capture gains.
- The crossover of EM+ and EM- can also be used as trend confirmation indicators.

Calculation

EMMinus is simply the percentage of bars that have achieved new lows within the specified lookback period. Consider, for example, the EMMinus with a period of 40. Within the past 40 bars there have been 10 bars that have reached a 40 bar low. The EMMinus indicator value for this bar would be 25, because 25% of the bars have reached new lows in the period.

Example

```
{ Enter short when EMMinus turns up and start a trailing
  stop when it crosses 19 }
var Bar, EMPane, hEMMinus: integer;
var TStopOn: boolean;
EMPane := CreatePane( 75, true, true );
hEMMinus := EMMinusSeries( #Close, 40 );
PlotSeriesLabel( hEMMinus, EMPane, 900, #Thin, 'EMMinus(#Close,40)' );

InstallStopLoss( 8 );
InstallBreakEvenStop( 5 );
PlotStops;
for Bar := 40 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );

  if LastPositionActive then
  begin
    if CrossOverValue( Bar, hEMMinus, 19 ) then
      TStopOn := true;

    if TStopOn then
      CoverAtTrailingStop( Bar, PriceHigh(Bar - 3) + 0.1,
                          LastPosition, 'TStop' );
  end
  else
  begin
    if ( @hEMMinus[Bar - 1] < 0.01 ) and TurnUp( Bar, hEMMinus ) then
      ShortAtMarket( Bar + 1, 'Bear' );
      TStopOn := false;
  end
end
```



```

end;
{ Looks like a bear trend has formed }
end;

```

16.24 EMPlus

EMPlus(Bar: integer; Series: integer; Period: integer): float;
 EMPlusSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

EM+ is the percentage of bars that have made new highs within the specified *Period*. See [EMMinus](#)^[189] for additional information.

Interpretation

- When EM+ rises from zero this is an indication that a worthwhile uptrend may be in the making. You can take a trend-following position at this point and exit once the indicator reaches a predetermined oversold level.
- Once EM+ crosses 20 prices tend to follow through and a profit target or trailing stop often works well to capture gains.
- The crossover of EM+ and EM- can also be used as trend confirmation indicators.

Calculation

EMPlus is simply the percentage of bars that have achieved new highs within the specified lookback period. Consider, for example, the EMPlus with a period of 40. Within the past 40 bars there have been 20 bars that have reached a 40 bar high. The EMPlus indicator value for this bar would be 50, because 50% of the bars have reached new highs in the period.

Example

```

var Bar, EMPane, EMPlus1: integer;
EMPane := CreatePane( 75, true, true );
EMPlus1 := EMPlusSeries( #Close, 40 );
PlotSeriesLabel( EMPlus1, EMPane, 009, #Thin,
'EMPlus1=EMPlus(#Close,40)' );

InstallTrailingStop( 2, 25 );
InstallStopLoss( 6 );
PlotStops;
for Bar := 40 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  { Looks like a bull trend has formed }
  if CrossOverValue( Bar, EMPlus1, 20 ) then
    BuyAtMarket( Bar + 1, 'Bull' );
end;

```

16.25 FAMA

FAMA(Bar: integer; Series: integer; FastLimit: float; SlowLimit: float): float;
 FAMASeries(Series: integer; FastLimit: float; SlowLimit: float): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

FAMA stands for Following Adaptive Moving Average. It was developed by John Ehlers of Mesa Software, and presented in the September 2001 issue of Stocks & Commodities magazine. FAMA is a complimentary indicator to MAMA (see MAMA indicator for more details).

The FAMA indicator uses an alpha (a) value that is half of its corresponding MAMA indicator. This results in an indicator that is synchronized to MAMA, but with vertical movement that is not as great. Consequently, MAMA and FAMA do not cross unless there has been a major change in market direction.

In addition to Price Series, FAMA accepts two additional parameters, FastLimit and SlowLimit. These control the maximum and minimum alpha (a) value that should be applied to the most recent bar of data when calculating FAMA.

You can learn more about the Mesa Adaptive Moving Average at the www.mesasoftware.com web site.

Interpretation

FAMA is used in conjunction with its complimentary MAMA indicator. Long signals occur when MAMA crosses above FAMA, and short signals when MAMA crosses below FAMA.

Calculation

$$\text{FAMA} = 0.5 * \text{alpha} * \text{MAMA} + (1 - 0.5 * \text{alpha}) * \text{Previous FAMA}$$

Example

```
var Bar: integer;
PlotSeries( MAMASeries( #Close, 0.5, 0.05 ), 0, #Red, #Thin );
PlotSeries( FAMASeries( #Close, 0.5, 0.05 ), 0, #Blue, #Thin );
for Bar := 40 to BarCount - 1 do
begin
  if CrossOver( Bar, MAMASeries( #Close, 0.5, 0.05 ),
               FAMASeries( #Close, 0.5, 0.05 ) ) then
    BuyAtMarket( Bar + 1, '' )
  else if CrossOver( Bar, FAMASeries( #Close, 0.5, 0.05 ),
                   MAMASeries( #Close, 0.5, 0.05 ) ) then
    SellAtMarket( Bar + 1, LastPosition, '' );
end;
```

16.26 FIR

```
FIR( Bar: integer; Series: integer; Filter: string ): float;
FIRSeries( Series: integer; Filter: string ): integer;
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

FIR stands for Finite Impulse Response Filter. This is a type of smoothing filter that assigns different weights to price data a number of bars in the past. Pass the Price Series you want to apply the filter to in the first parameter. The second parameter of the FIR is a string that describes the weights that will be applied to the bars of data that compose the filter. The string is formatted as a series of numbers separated by commas.

Interpretation

FIR filters are nothing more than another type of weighted moving average, with

different weight levels applied to the various components of the average. As such, you can apply any of the various interpretations of moving averages to FIR.

Calculation

A simple example will make this concept easier to explain. Assume we pass the string value of '4,3,2,1' as the second parameter to FIR, and apply it to closing prices. The function will perform the following calculation:

$$\frac{((4 \times \text{current Closing Price}) + (3 * \text{Closing Price 1 bar back}) + (2 * \text{Closing Price 2 bars back}) + (1 * \text{Closing Price 3 bars back}))}{10}$$

As you can see, each successive weight value is applied to the previous bar back in the price history. The final sum of the weighted price values is divided by the sum of the weights.

Example

```
{ A FIR is used as a signal line for a 200 day moving average }
var SMASer: integer;
SMASer := SMASeries( #Close, 200 );
PlotSeries( SMASer, 0, #Olive, #Thick );
PlotSeries( FIRSeries( SMASer, '1,2,2,1' ), 0, #Black, #Thin );
```

16.27 Highest

Highest(Bar: integer; Series: integer; Period: integer): float;
HighestSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the highest value of a Price Series within the specified look back *Period*.

Highs and lows are found in all markets, and, they are fundamentally important to technical analysis. Uptrends are defined by a succession of higher highs and higher lows, whereas downtrends are characterized by a succession of lower highs and lower lows. Support and resistance is often defined at significant high and low points. Pattern formations are shaped by highs and lows such as double tops and bottoms, head and shoulder formations, rectangles, triangles, flags, consolidations and other formations. Use Wealth-Lab functions: **Highest**, **Lowest**, **HighestBar** and **LowestBar**, for finding and analyzing these market opportunities.

Interpretation

- Uptrends are formed by a succession of higher highs and higher lows, failure to make a new higher high or higher low means the trend has ended.
- Downtrends are formed by a succession of lower highs and lower lows failure to make a new lower high or lower low means the trend has ended.
- Divergence occurs when price action and indicators move in different directions and commonly occur before a stock or a market changes direction. During an uptrend, if a new high appears in the price action and a lower high develops in the indicator, then the trend maybe ending. See RSI for more information on divergence.
- Useful for stops loss calculations such as when taking short positions, or buy order triggers. Add an amount to the last highest value, this can be a fixed amount like a percentage of current closing price. A better method is to allow for volatility, use a multiple of ATR for this. For example, Highest high plus, half times ATR of last 20

bars.

- Entry filters, buy if today prices is higher than high for past two days. This can be useful to ensure you start your trade in the right direction.
- Past highs can represents significant resistance to price action. Look for single or multiple highs forming at both technical and psychological levels. (Gann, Fibonacci, whole numbers, and the like).
- Many chart patterns are defined by recent high and lows.

Calculation

Looks back the specified number of periods from the specified Bar and returns the highest price within that *Period*.

Example

```
{ Have we made a 100 bar high? }
var BAR: integer;
for Bar := 100 to BarCount - 1 do
  if PriceHigh( Bar ) = Highest( Bar, #High, 100 ) then
    AnnotateBar( 'NH', Bar, true, #Black, 8 );
```

16.28 HighestBar

HighestBar(Bar: integer; Series: integer; Period: integer): integer;
HighestBarSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the bar in which highest value of the Price Series for the specified *Period* was recorded. Refer to [Highest](#)^[192] for more information.

Interpretation

(See [Highest](#)^[192])

Remarks

- If more than one bar has precisely the same **Highest** value, then **HighestBar** returns the *most recent* bar, i.e., the bar with the latest date/time.

Calculation

Looks back the specified number of periods from the specified Bar and returns the Bar number with the highest price within that period.

Example

```
{ Has the 200 day high ocurred within the past 20 bars? }
var N: float;
var BAR: integer;
for Bar := 200 to BarCount - 1 do
begin
  n := HighestBar( Bar, #High, 200 );
  if Bar - n <= 20 then
    SetBackgroundColor( Bar, 888 );
end;
```

16.29 HTDCPhase

HTDCPhase(Bar: integer; Series: integer): float;
HTDCPhaseSeries(Series: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Hilbert Transform is a technique used to generate Inphase and Quadrature components of a de-trended real-valued "analytic-like" signal (such as a Price Series) in order to analyze variations of the instantaneous phase and amplitude.. HTDCPhase returns the Hilbert Transform Phase of the Dominant Cycle. The Dominant Cycle Phase lies in the range of 0 to 360 degrees.

Interpretation

The DC Phase at a specific bar gives the phase position from 0 to 360 degrees within the current Hilbert Transform Period instantaneously measured at that bar. It is meaningful only during a cyclic period of the analytic signal waveform (price series) being measured. Its transition from 360 degrees to 0 degrees can be used to designate the start of a new cycle. It can also be utilized to signal the start or end of trending or cyclic periods. Departure from a constant rate change of phase is a sensitive way to detect the end of a cycle mode. See the examples.

Calculation

More detailed information concerning the calculation of the Hilbert Transform related functions can be found in this document and others on the Mesa Software site:

<http://www.mesasoftware.com/pub/concepts.exe>

The basic flow and simplified pseudo code for the computation for the Dominant Cycle Phase as part of the computation of the Dominant Cycle is:

Compute the Hilbert Transform

{Detrend Price}
{Compute InPhase and Quadrature components}

Compute the Period of the Dominant Cycle

{Use ArcTangent to compute the current phase}
{Resolve the ArcTangent ambiguity}

{Compute a differential phase, resolve phase wraparound, and limit delta phase errors}

{Sum DeltaPhases to reach 360 degrees. The sum is the instantaneous period.}

{Resolve Instantaneous Period errors and smooth}

Compute Dominant Cycle Phase

Return the Dominant Cycle Phase at the current bar of the Hilbert Transform Period measured at that bar

Example

```
{ Example - Make bars more solid as the dominant cycle phase approaches
360 }
var DCPHASEPANE, htDCP, BAR, N: integer;

DCPhasePane := CreatePane( 100, true, true );
htDCP := HTDCPhaseSeries( #Average );
PlotSeries( htDCP, DCPHASEPANE, 520, #Thick );
for Bar := 0 to BarCount - 1 do
begin
  n := 9 - ( Round( @htDCP[Bar] / 360 * 9 ) );
```

```

    SetBarColor( Bar, n * 100 + n * 10 + n );
end;

{ Also see ChartScript 'RocketScience v1' }

```

16.30 HTInPhase

HTInPhase(Bar: integer; Series: integer): float;
 HTInPhaseSeries(Series: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Hilbert Transform is a technique used to generate inphase and quadrature components of a de-trended real-valued "analytic-like" signal (such as a Price Series) in order to analyze variations of the instantaneous phase and amplitude. HTInPhase returns the Hilbert Transform generated InPhase component of the input Price Series.

Interpretation

The InPhase component is used in conjunction with the Quadrature component to generate the phase of the analytic signal (using the ArcTan function) at a specific bar or for the entire Price Series.

Calculation

More detailed information concerning the calculation of the Hilbert Transform related functions can be found in this document and others on the Mesa Software site:

<http://www.mesasoftware.com/pub/concepts.exe>

The basic flow for the computation for the InPhase component is:

Compute the Hilbert Transform

```

    {Detrend Price}
    {Compute InPhase and Quadrature components}

```

Return the InPhase component at the current bar of the Hilbert Transform computed at that bar

Example

```

{ Color bars based on relative positions of Quadrature and In-Phase }
var HTQUADPANE, BAR: integer;
HTQuadPane := CreatePane( 100, true, true );
PlotSeries( HTInPhaseSeries( #Average ), HTQuadPane, 025, #Thick );
PlotSeries( HTQuadratureSeries( #Average ), HTQuadPane, 559, #Thick );
for Bar := 0 to BarCount - 1 do
begin
  if HTQuadrature( Bar, #Average ) > HTInPhase( Bar, #Average ) then
    SetBarColor( Bar, 559 )
  else
    SetBarColor( Bar, 025 );
end;

```

16.31 HTLeadSin

HTLeadSin(Bar: integer; Series: integer): float;
 HTLeadSinSeries(Series: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Hilbert Transform is a technique used to generate inphase and quadrature components of a de-trended real-valued "analytic-like" signal (such as a Price Series) in order to analyze variations of the instantaneous phase and amplitude.. HTDCPhase returns the Hilbert Transform Phase of the Dominant Cycle. The Dominant Cycle Phase lies in the range of 0 to 360 degrees. The Hilbert Transform Lead Sine is just the sine of the DC Phase advanced by 45 degrees.

Interpretation

The DC Phase at a specific bar gives the phase position from 0 to 360 degrees within the current Hilbert Transform Period instantaneously measured at that bar. The HTLeadSin is the sine of the DC Phase at a specific bar. It is most often used in conjunction with the HTSin indicator to identify cyclic turning points. Quoting from Market Mode Strategies.doc by John Ehlers from MESA Software, "A clear, unequivocal cycle mode indicator can be generated by plotting the Sine of the measured phase angle advanced by 45 degrees. This leading signal crosses the sinewave 1/8th of a cycle BEFORE the peaks and valleys of the cyclic turning points, enabling you to make your trading decision in time to profit from the entire amplitude swing of the cycle. A significant additional advantage is that the two indicator lines don't cross except at cyclic turning points, avoiding the false whipsaw signals of most "oscillators" when the market is in a Trend Mode. The two lines don't cross because the phase rate of change is nearly zero in a trend mode. Since the phase is not changing, the two lines separated by 45 degrees in phase never get the opportunity to cross." See the examples.

Calculation

More detailed information concerning the calculation of the Hilbert Transform related functions can be found in this document and others on the Mesa Software site:

<http://www.mesasoftware.com/pub/concepts.exe>

The basic flow and simplified pseudo code for the computation for the HTLeadSin is:

Compute the Hilbert Transform

{Detrend Price}

{Compute InPhase and Quadrature components}

Compute the Period of the Dominant Cycle

{Use ArcTangent to compute the current phase}

{Resolve the ArcTangent ambiguity}

{Compute a differential phase, resolve phase wraparound, and limit delta phase errors}

{Sum DeltaPhases to reach 360 degrees. The sum is the instantaneous period.}

{Resolve Instantaneous Period errors and smooth}

Compute Dominant Cycle Phase

Compute the Sine of the Dominant Cycle Phase

Advance the Sine by 45 degrees to compute the HT Lead Sine

Return the Lead Sine of the Dominant Cycle Phase at the current bar of the Hilbert Transform Period measured at that bar

Example

```
{ Flag bars where Hilbert Transform Sin/Lead Sin cross }
var HTSINPANE, BAR, HTLead, HT: integer;
HT := HTSinSeries( #Average );
HTLead := HTLeadSinSeries( #Average );
HTSinPane := CreatePane( 100, false, true );
PlotSeries( HTLead, HTSinPane, 009, #Thin );
PlotSeries( HT, HTSinPane, 900, #Thin );
```

```

for Bar := 2 to BarCount - 1 do
begin
  if CrossOver( Bar, HT, HTLead ) then
    SetBarColor( Bar, #Red )
  else if CrossUnder( Bar, HT, HTLead ) then
    SetBarColor( Bar, #Blue );
end;

```

16.32 HTPeriod

HTPeriod(Bar: integer; Series: integer): float;
 HTPeriodSeries(Series: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Hilbert Transform is a technique used to generate Inphase and Quadrature components of a de-trended real-valued "analytic-like" signal (such as a Price Series) in order to analyze variations of the instantaneous phase and amplitude. HTPeriod (or MESA Instantaneous Period) returns the period of the Dominant Cycle of the analytic signal as generated by the Hilbert Transform. The Dominant Cycle can be thought of as being the "most likely" period (in the range of 10 to 40) of a sine function of the Price Series.

Interpretation

The HTPeriod at a specific bar gives the current Hilbert Transform Period as instantaneously measured at that bar in the range of 10 to 40. It is meaningful only during a cyclic period of the analytic signal waveform (price series) being measured. The HTPeriod, or one of its sub-periods, is often used to adjust other indicators; for example, Stochastics and RSIs work best when using a half cycle period to peak their performance. Similarly other indicators can be made to be adaptive by using the HTPeriod, or one of its sub-periods, as the period of the indicator. See the examples.

Calculation

More detailed information concerning the calculation of the Hilbert Transform related functions can be found in this document and others on the Mesa Software site:

<http://www.mesasoftware.com/pub/concepts.exe>

The basic flow and simplified pseudo code for the computation for the Dominant Cycle Period is:

```

Compute the Hilbert Transform
  {Detrend Price}
  {Compute InPhase and Quadrature components}
Compute the Period of the Dominant Cycle
  {Use ArcTangent to compute the current phase}
  {Resolve the ArcTangent ambiguity}
  {Compute a differential phase, resolve phase wraparound, and limit delta
phase errors}
  {Sum DeltaPhases to reach 360 degrees. The sum is the instantaneous
period.}
  {Resolve Instantaneous Period errors and smooth}
Return the Hilbert Transform Period measured at the current bar

```


Example

```

{ This code creates an adaptive moving average.
  The period of the MA is based on the HTPeriod for the bar }
var HTPERIODPANE, DYN SMA, BAR, N: integer;
HTPeriodPANE := CreatePane( 50, true, true );
PlotSeries( HTPeriodSeries( #Average ), HTPeriodPANE, 161, #Thick );
DrawLabel( 'HTPeriod( Average )', HTPeriodPANE );
DynSMA := CreateSeries;
for Bar := 40 to BarCount - 1 do
begin
  n := Round( HTPeriod( Bar, #Average ) );
  if n < 2 then
  n := 2;
  SetSeriesValue( Bar, DynSMA, SMA( Bar, #Average, n ) );
end;
PlotSeries( DynSMA, 0, #Navy, #Thick );

{ Example - An alternative method of computing the HT Period using a
Homodyne Discriminator can be used which demonstrates different
sensitivity than the Phase Accumulation approach. See ChartScript -
MesaPeriodCheck V2, http://www.wealth-lab.com/cgi-bin/WealthLab.DLL/editsystem?id=4805 by ttcrep for a comparison of the
two waveforms}

{ Example - Here the HTPeriod is used to make the acceleration of a
Parabolic SAR adaptive. From Parabolic SAR CyclePeriod,
http://www.wealth-lab.com/cgi-bin/WealthLab.DLL/editsystem?id=2815, by
Willibald. Adapted and declarations and most of script omitted.}

```

16.33 HTQuadrature

```

HTQuadrature( Bar: integer; Series: integer ): float;
HTQuadratureSeries( Series: integer ): integer;

```

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Hilbert Transform is a technique used to generate inphase and quadrature components of a de-trended real-valued "analytic-like" signal (such as a Price Series) in order to analyze variations of the instantaneous phase and amplitude. HTQuadrature returns the Hilbert Transform generated Quadrature component of the input Price Series.

Interpretation

The Quadrature component is used in conjunction with the InPhase component to generate the phase of the analytic signal (using the ArcTan function) at a specific bar or for the entire Price Series.

Calculation

More detailed information concerning the calculation of the Hilbert Transform related functions can be found in this document and others on the Mesa Software site:

<http://www.mesasoftware.com/pub/concepts.exe>

The basic flow for the computation for the InPhase component is:

Compute the Hilbert Transform

{Detrend Price}

{Compute InPhase and Quadrature components}

Return the Quadrature component at the current bar of the Hilbert Transform

computed at that bar

Example

```
{ Example - Use HTInPhase and HTQuadrature to compute
  the Signal-to-Noise Ratio (SNR) for a Price Series }
var X1, X2, X4, X3, X5: float;
var RANGE, TEMP1, TEMP2, AMPLITUDE, BAR, NPANE: integer;
Range := CreateSeries();
Temp1 := CreateSeries();
Temp2 := CreateSeries();
Amplitude := CreateSeries();

FOR Bar := 2 to BarCount - 1 do
BEGIN
  x1 := HTInPhase( Bar, #Average );
  x2 := HTQuadrature( Bar, #Average );
  {Compute "Noise" as average range. x4 = Current Bar range}
  x4 := 0.1 * (PriceHigh(Bar) - PriceLow(Bar)) + (0.9 *
GetSeriesValue(Bar - 1, Range));
  SetSeriesValue(Bar, Range, x4);
  {Compute smoothed signal amplitude - x3 = Current Bar Temp1}
  x3 := (0.2 * ((x1 * x1) + (x2 * x2))) + (0.8 * GetSeriesValue(Bar -
1, Temp1));
  IF x3 < 0.001 THEN
    x3 := 0.001;
  SetSeriesValue(Bar, Temp1, x3);
  {Compute smoothed SNR in dB guarding against divide by zero}
  IF x4 > 0.0 THEN
    x5 := 0.25 * ( 10.0 * Log10(x3 / (x4 * x4)) + 1.9 ) + 0.75 *
GetSeriesValue(Bar - 1, Temp2);
    SetSeriesValue(Bar, Temp2, x5);
    SetSeriesValue(Bar, Amplitude, x5);
  END;

  {Plot SNR in dB}
  nPane := CreatePane(150, true, true);
  PlotSeries(Amplitude, nPane, #Red, 0);
```

16.34 HTSin

HTSin(Bar: integer; Series: integer): float;
HTSinSeries(Series: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Hilbert Transform is a technique used to generate inphase and quadrature components of a de-trended real-valued "analytic-like" signal (such as a Price Series) in order to analyze variations of the instantaneous phase and amplitude.. HTDCPhase returns the Hilbert Transform Phase of the Dominant Cycle. The Dominant Cycle Phase lies in the range of 0 to 360 degrees. The Hilbert Transform Sine is just the sine of the DC Phase.

Interpretation

The DC Phase at a specific bar gives the phase position from 0 to 360 degrees within the current Hilbert Transform Period instantaneously measured at that bar. The HTSin is the sine of the DC Phase at a specific bar. It is most often used in conjunction with the HTLeadSin indicator to identify cyclic turning points. Quoting from Market Mode Strategies.doc by John Ehlers from MESA Software, "A clear, unequivocal cycle mode

indicator can be generated by plotting the Sine of the measured phase angle advanced by 45 degrees. This leading signal crosses the sinewave 1/8th of a cycle BEFORE the peaks and valleys of the cyclic turning points, enabling you to make your trading decision in time to profit from the entire amplitude swing of the cycle. A significant additional advantage is that the two indicator lines don't cross except at cyclic turning points, avoiding the false whipsaw signals of most "oscillators" when the market is in a Trend Mode. The two lines don't cross because the phase rate of change is nearly zero in a trend mode. Since the phase is not changing, the two lines separated by 45 degrees in phase never get the opportunity to cross." See the examples.

Calculation

More detailed information concerning the calculation of the Hilbert Transform related functions can be found in this document and others on the Mesa Software site: <http://www.mesasoftware.com/pub/concepts.exe>

The basic flow and simplified pseudo code for the computation for the Dominant Cycle Phase as part of the computation of the Dominant Cycle is:

```

Compute the Hilbert Transform
    {Detrend Price}
    {Compute InPhase and Quadrature components}
Compute the Period of the Dominant Cycle
    {Use ArcTangent to compute the current phase}
    {Resolve the ArcTangent ambiguity}
    {Compute a differential phase, resolve phase wraparound, and limit delta
phase errors}
    {Sum DeltaPhases to reach 360 degrees. The sum is the instantaneous
period.}
    {Resolve Instantaneous Period errors and smooth}
Compute Dominant Cycle Phase
Compute the Sine of the Dominant Cycle Phase
Return the Sine of the Dominant Cycle Phase at the current bar of the Hilbert
Transform Period measured at that bar

```

Example

```

{ Flag bars where Hilbert Transform Sin/Lead Sin cross }
var HTSINPANE, HTSINser, BAR: integer;
HTSINser := HTSinSeries( #Average );
HTSinPane := CreatePane( 100, false, true );
PlotSeries( HTSINser, HTSinPane, 009, #Thin );
PlotSeries( HTLeadSinSeries( #Average ), HTSinPane, 900, #Thin );
for Bar := 2 to BarCount - 1 do
begin
    if CrossOver( Bar, HTSINser, HTLeadSinSeries( #Average ) ) then
        SetBarColor( Bar, #Red )
    else if CrossUnder( Bar, HTSINser, HTLeadSinSeries( #Average ) ) then
        SetBarColor( Bar, #Blue );
end;

```

16.35 HTTrendLine

```

HTTrendLine( Bar: integer; Series: integer ): float;
HTTrendLineSeries( Series: integer ): integer;

```

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Hilbert Transform is a technique used to generate inphase and quadrature components of a de-trended real-valued "analytic-like" signal (such as a Price Series) in order to analyze variations of the instantaneous phase and amplitude. HTTrendline (or MESA Instantaneous Trendline) returns the Price Series value after the Dominant Cycle of the analytic signal as generated by the Hilbert Transform has been removed. The Dominant Cycle can be thought of as being the "most likely" period (in the range of 10 to 40) of a sine function of the Price Series.

Interpretation

The HTTrendline at a specific bar gives the current Hilbert Transform Trendline as instantaneously measured at that bar. In its Series form, the Instantaneous Trendline appears much like a Moving Average, but with minimal lag compared with the lag normally associated with such averages for equivalent periods. The HTTrendline is formed by removing the Dominant Cycle from the Price Series. See the examples.

Calculation

More detailed information concerning the calculation of the Hilbert Transform related functions can be found in this document and others on the Mesa Software site: <http://www.mesaSoftware.com/pub/concepts.exe>. Quoting from MarketMode Strategies.doc, "A simple average taken over the period of the dominant cycle has as many sample points above the average as below it, with the result that the dominant cycle component is removed at the output of the filter. The filtered residual is the Instantaneous Trendline." The basic flow and simplified pseudo code for the computation for the Dominant Cycle Period is:

```

Compute the Hilbert Transform
    {Detrend Price}
    {Compute InPhase and Quadrature components}
Compute the Period of the Dominant Cycle
    {Use ArcTangent to compute the current phase}
    {Resolve the ArcTangent ambiguity}
    {Compute a differential phase, resolve phase wraparound, and limit delta
phase errors}
    {Sum DeltaPhases to reach 360 degrees. The sum is the instantaneous
period.}
    {Resolve Instantaneous Period errors and smooth}
Compute the Instantaneous Trendline
    {Average over the period of the Dominant Cycle at each bar}
Return the Hilbert Transform Trendline measured at the current bar

```

Example

```

{ We focus here on the relationship between the Instantaneous Trendline
and a Zero Lag Kalman Filter. Quoting from Tutorial.doc by John Ehlers
of MESA Software: "If we use a Zero Lag Kalman Filter, this filter
line will cross the Instantaneous Trendline every half cycle when the
market is in a Cycle Mode. If the Zero Lag Kalman filter fails to
cross the Instantaneous Trendline within the last half cycle period,
then this is another way of declaring a Trend Mode is in force. The
Trend Mode ends when the Zero Lag Kalman Filter line again crosses the
Instantaneous Trendline." }
var PERIODPANE: integer;
PlotSeries( HTTrendLineSeries( #Average ), 0, 732, #Thick );
PlotSeries( KalmanSeries( #Average ), 0, 000, #Dotted );
PeriodPane := CreatePane(80, true, true);

```

```
PlotSeries(HTPeriodSeries( #Average ), PeriodPane, #Red, #Thick );
```

16.36 HV

HV(Bar: integer; Series: integer; Period: integer; Span: integer): float;
 HVSeries(Series: integer; Period: integer; Span: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Historical Volatility of the selected Price Series. Historical Volatility is the standard deviation of the logarithm of the price ratio, i.e.

$$HV = \text{Standard Deviation} (\ln(\text{Price}(\text{Bar}) / \text{Price}(\text{Bar}-1)))$$

Series Specifies which Price Series to use, for example **#Close**, **#Average**, etc. *Series* can also be a technical indicator series, or a Price Series created within the script.

Period e.g. 20, specifies how many bars **HV** shall use. The real number of periods that **HV** will use is *Period* - 1, because if for example you use 20 price bars, there are 19 periods in between and 19 returns.

Span Used to convert the historical volatility to a different time scale. If the Chart has weekly bars and annualized historical volatility is required, use 52 for *Span* because there are 52 weeks in a year.

Interpretation

A sharp increase in HV will alert you to unusual volatility in the markets. This is often an ideal time to monitor the market for entry in the opposite direction of the panic.

Calculation

$$HV = \text{Sqrt}(\text{SSD} / (\text{Period} - 1)) * \text{Sqrt}(\text{Span})$$

where,

$$\text{SSD} = \text{Sum}[(\text{LOGSi} - \text{ALOGS})^2] \text{ over Period bars}$$

LOGSi = Logarithm of Price - Previous Price

ALOGS = Sum(Logarithms of Price Change over Span) / Span

Example

```
var Bar: integer;
var HVPane: integer;
HVPane := CreatePane( 75, true, true );
var HV1: integer;
HV1 := HVSeries( #Average, 20, 262 );
PlotSeriesLabel( HV1, HVPane, 905, #Thick, 'HV1=HV(#Average,20,262)' );
InstallProfitTarget( 15 );
InstallStopLoss( 40 );
for Bar := 262 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    if @HV1[Bar] > 100 then
    begin
      if ROC( Bar, #Close, 30 ) > 0 then
        ShortAtMarket( Bar + 1, '' )
```

```

        else
            BuyAtMarket( Bar + 1, '' );
        end;
    end
else
    ApplyAutoStops( Bar );
end;

```

16.37 Kalman

Kalman(Bar: integer; Series: integer): float;
 KalmanSeries(Series: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Kalman Filter is based on the concept of optimum estimation, first introduced by Dr. R. E. Kalman in 1960. It has generally been used in terrestrial and space-based navigation and tracking systems. In general, it can be thought of as generating an optimal (in a linear, white noise, mean-square-error sense) estimate of a future position based on the current position of a target and an estimate of its velocity and acceleration and their uncertainties.

Interpretation

Where Price Series are involved as in trading systems the mathematics can be simplified considerably and a (nearly) zero lag filter produced very straightforwardly. For further information see: Optimal Tracking Filters.doc by John Ehlers of MESA Software, here: <http://www.mesaSoftware.com/pub/TRACKINGFILTERS.EXE>. Note that Kalman filters can be applied to any Price Series, not just ticker prices. See Examples.

Calculation

The basic pseudo computation for the Kalman Filter value at a specific bar for a Price Series is:

```

ZeroLagValue at Bar = Weight1 * PriceSeriesValue at Bar + Weight2
                    (PriceSeriesValue at Bar - PriceSeriesValue at
                     Bar - 3)

```

```

ZeroLagValue at Bar = ZeroLagValue at Bar + Weight3 * LastZeroLagValue

```

```

LastZeroLagValue = ZeroLagValue at Bar

```

```

Save KalmanSeriesValue at Bar = ZeroLagValue at Bar

```

```

Return KalmanSeriesValue at Bar

```

Example

```

{ This system uses the Kalman Filter as a signal line for the
  CMO Oscillator to time position entries }
var CMOPANE, BAR, hCMO: integer;
hCMO := CMOSeries( #Average, 14 );
CMOPane := CreatePane( 80, true, true );
PlotSeries( hCMO, CMOPane, 009, #Thick );
PlotSeries( KalmanSeries( hCMO ), CMOPane, #Black, #Thin );
for Bar := 14 to BarCount - 1 do
begin
    if LastPositionActive then
    begin
        if CrossOverValue( Bar, hCMO, 0 ) then

```

```

        SellAtMarket( Bar + 1, LastPosition, 'CMO 0' );
    end
    else
    begin
        if CMO( Bar - 1, #Average, 14 ) < -50 then
            if CrossOver( Bar, hCMO, KalmanSeries( hCMO ) ) then
                BuyAtMarket( Bar + 1, 'CMO Kalman' );
            end;
        end;
    end;
end;

```

16.38 KAMA

KAMA(Bar: integer; Series: integer; Period: integer): float;
 KAMASeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns Kaufman's Adaptive Moving Average for the Price Series specified in the **Series** parameter. KAMA is an adaptive moving average, and uses the noise level of the market to determine the length of the trend required to calculate the average. The more noise in the market, the slower the trend used to calculate the average. The **Period** parameter control how much data is used by KAMA to calculate its efficiency ratio (signal/noise). A value of 8 to 10 is recommended.

Interpretation

- You can base trading signals on whether KAMA turns down or up, indicating a potential trend reversal. Kaufman suggests using a small band around the KAMA indicator as a way to filter out whipsaws.
- Since KAMA is a type of moving average, you can use the same interpretation techniques used for Simple Moving Averages (SMA).

Calculation

```

{ The WealthScript code below duplicates the KAMA indicator
calculation: }
var AMA, SIGNAL, DIFF, NOISE, EFRATIO, SMOOTH: float;
var MYKAMA, BAR, J, Period: integer;

Period := 10;
MyKAMA := CreateSeries;
{ initialize the starting period, ama }
for Bar := 0 to Period do
    @MyKAMA[Bar] := PriceClose(Bar);
ama := PriceClose( Period );

for Bar := Period + 1 to BarCount - 1 do
begin
    signal := Abs( PriceClose( Bar ) - PriceClose( Bar - Period ) );
    noise := 0;
    for j := 0 to Period - 1 do
    begin
        diff := Abs( PriceClose( Bar - j ) - PriceClose( Bar - j - 1 ) );
        noise := noise + diff;
    end;
    if noise <> 0 then
        efratio := signal / noise
    else
        efratio := 0;
    smooth := efratio * ( 2 / 3 - 2 / 31 ) + 2 / 31;
end;

```

```

smooth := smooth * smooth;
ama := ama + smooth * ( PriceClose( Bar ) - ama );
@MyKAMA[Bar] := ama;
end;
PlotSeries( MyKAMA, 0, #Red, #Histogram );
PlotSeries( KAMASeries( #Close, Period ), 0, #Black, #Thin );

```

Example

```

var K1, K2, BAR: integer;
K1 := KAMASeries( #Close, 8 );
K2 := KAMASeries( #Close, 16 );
PlotSeriesLabel( K1, 0, #Red, #Thin, 'KAMA(8)' );
PlotSeriesLabel( K2, 0, #Maroon, #Thin, 'KAMA(16)' );
for Bar := 18 to BarCount - 1 do
begin
  if LastPositionActive and CrossUnder( Bar, K1, K2 ) then
    SellAtMarket( Bar + 1, LastPosition, '' )
  else if CrossOver( Bar, K1, K2 ) then
    BuyAtMarket( Bar + 1, '' );
end;

```

16.39 KeltnerLower

KeltnerLower(Bar: integer; Period1: integer; Period2: integer): float;
 KeltnerLowerSeries(Period1: integer; Period2: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Keltner Bands are a type of price channel first described by Chester W. Keltner in his book *How to Make Money in Commodities*. They are fixed bands that are plotted above and below a simple moving average of average price.

The Keltner indicators in Wealth-Lab take two parameters. *Period1* specifies the period to smooth highs - lows, and *Period2* specifies the period to use to smooth Average Price in the calculation (see below). Note that because Keltner Bands are defined to use average price, and highs minus lows, the indicator does not take a Price Series parameter like many other indicator functions.

Interpretation

- The classic interpretation of Keltner band is to go long when the upper band is penetrated, and reverse position and enter short when the lower band is penetrated.
- Keltner Bands can also be used to define "normal" trading ranges for markets. Price movement outside of the bands can be considered an anomaly, and therefore a trading opportunity.

Calculation

Average Price (AP) = (Close + High + Low) / 3

Band Moving Average = *Period1* bar Simple Moving Average (SMA) of (High - Low)

Center Line = *Period2* bar SMA of AP

Upper Band = Center Line + Band MA

Lower Band = Center Line - Band MA

Example

```

var BAR: integer;
PlotSeries( KeltnerLowerSeries( 10, 10 ), 0, 151, #Thick );
PlotSeries( KeltnerUpperSeries( 10, 10 ), 0, 151, #Thick );
for Bar := 30 to BarCount - 1 do
begin
  if CrossOver( Bar, #Close, KeltnerUpperSeries( 10, 10 ) ) then
  begin
    if not PositionLong( LastPosition ) then
    begin
      CoverAtMarket( Bar + 1, LastPosition, '' );
      BuyAtMarket( Bar + 1, '' );
    end;
  end
  else if CrossUnder( Bar, #Close, KeltnerLowerSeries( 10, 10 ) ) then
  begin
    if PositionLong( LastPosition ) then
    begin
      SellAtMarket( Bar + 1, LastPosition, '' );
      ShortAtMarket( Bar + 1, '' );
    end;
  end;
end;
end;

```

16.40 KeltnerUpper

KeltnerUpper(Bar: integer; Period1: integer; Period2: integer): float;
 KeltnerUpperSeries(Period1: integer; Period2: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

See [KeltnerLower](#)^[205]

16.41 LinearReg

LinearReg(Bar: integer; Series: integer; Period: integer): float;
 LinearRegSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Linear Regression value for the specified *Series* and *Period*. **LinearReg** gathers the prices for the number of specified periods and finds a straight line (see **LinearRegLine**) which best fits all the prices using a linear regression model. The procedure is reinitialized and repeated for each bar in the Price Series. Since a new value is calculated at each *Bar*, the result is not a straight linear regression trendline; rather, it is an indicator which loosely tracks the price action.

LinearReg is a statistical indicator. Other indicators in the same class are **LinearRegSlope**, **StdErr**, **RSquared** and **StdDev**.

Interpretation

- Since the Linear Regression indicator displays the statistically-predicted price value, you can look for cases where price veers sharply from the predicted value. Use **RSquared** to determine significant weakness in the trend and if due for a return to the predicted value.

Calculation

Linear Regression is a rather complex statistical calculation. It uses the least square method to fit a trendline to the data by minimizing the distance between the price and the Linear Regression trendline. **LinearReg** returns the final value of the **LinearRegLine**, recalculated for each bar over the regression *Period* to complete indicator.

Example

```
{ Report on how far closing prices are away from predicted value }
var S: string;
var DIFF: float;
var BAR: integer;
PlotSeries( LinearRegSeries( #Close, 20 ), 0, 002, #Thick );
Bar := BarCount - 1;
Diff := PriceClose( Bar ) - LinearReg( Bar, #Close, 20 );
Diff := Diff / PriceClose( Bar ) * 100;
if Diff > 0 then
    s := 'Price closed above '
else
    s := 'Price closed below ' ;
Diff := Abs( Diff );
s := s + 'the Regression Line by ' + FormatFloat( '#0.00%', Diff );
DrawLabel( s, 0 );
```

16.42 LinearRegPredict

LinearRegPredict(Bar: integer; Series: integer): float;
LinearRegPredictSeries(Series: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Calculates a linear regression on a complete Price *Series* and returns the predicted value at a specific *Bar*.

Remarks:

- **LinearRegPredict** is designed for use in scriptable Performance Reports.
- Do not use the result of **LinearRegPredict** in a ChartScript for trading system rules.

Example

```
var h: integer;

h := LinearRegPredictSeries( #Close );
PlotSeries( h, 0, #Blue, #Thick );
```

16.43 LinearRegSlope

LinearRegSlope(Bar: integer; Series: integer; Period: integer): float;
LinearRegSlopeSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Linear Regression Slope returns the slope of the Linear Regression line (see

LinearRegLine) of the specified period. It looks at the prices for the number of specified periods and finds a straight line which best fits all the prices. The slope of this straight line is returned. Use the slope to determine if the trend is up (positive value) or down (negative value), as well as the general strength of the trend. It shows how much the prices are expected to change over time.

Linear Regression Slope indicator is a statistical indicator. Other indicators in the same class are **LinearReg**, **StdErr**, **RSquared** and **StdDev**.

Interpretation

- An up-sloping Linear Regression line ($\text{LinearRegSlope} > 0$) indicates that prices have been rising within the regression period, you could open a long position if the rising trend is significant. Use **RSquared** to determine trend significances.
- A down-sloping line ($\text{LinearRegSlope} < 0$) indicates prices have been falling within the regression period, you could open a short position if the decline is significant. Use **RSquared** to determine trend significances.
- You can open a contrary short-term position to the prevailing trend when the Linear Regression Slope begins to round off at extreme levels.

Calculation

Linear Regression is a rather complex statistical calculation. It uses the least square method to fit a trendline to the data by minimizing the distance between the price and the Linear Regression trendline. The slope of this Linear Regression trendline (given by **LinearRegLine**) is the value return by the **LinearRegSlope** indicator.

Example

```
{ Minor up and down trends highlighted by confirmation of 2 linear
regression lines }
var Bar: integer;
var LinRegSlopePane: integer;
LinRegSlopePane := CreatePane( 100, true, true );
PlotSeries( LinearRegSlopeSeries( #Close, 20 ), LinRegSlopePane, 205,
#Thin );
DrawLabel( 'LinearRegSlope( Close, 20 )', LinRegSlopePane );
PlotSeries( LinearRegSlopeSeries( #Close, 10 ), LinRegSlopePane, 509,
#Thin );
DrawLabel( 'LinearRegSlope( Close, 10 )', LinRegSlopePane );
for Bar := 20 to BarCount - 1 do
begin
    if LinearRegSlope( Bar, #Close, 20 ) > 0 then
        if LinearRegSlope( Bar, #Close, 10 ) > 0 then
            SetBarColor( Bar, #Blue );
    if LinearRegSlope( Bar, #Close, 20 ) < 0 then
        if LinearRegSlope( Bar, #Close, 10 ) < 0 then
            SetBarColor( Bar, #Red );
end;
```

16.44 Lowest

Lowest(Bar: integer; Series: integer; Period: integer): float;
 LowestSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the lowest value of a Price Series within the specified look back *Period*.

See [Highest](#)¹⁹² for more information.

Calculation

Looks back the specified number of periods from the specified Bar and returns the lowest price within that *Period*.

Example

```
{ Plot the most recent 40 bar low as dots on the chart }
PlotSeries( LowestSeries( #Low, 40 ), 0, #Maroon, #Dots );
DrawLabel( 'Lowest( Low, 40 )', 0 );
```

16.45 LowestBar

LowestBar(Bar: integer; Series: integer; Period: integer): integer;
LowestBarSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the bar in which lowest value of the Price Series for the specified *Period* was recorded.

See [Highest](#)¹⁹² for more information.

Remarks

- If more than one bar has precisely the same **Lowest** value, then **LowestBar** returns the *most recent* bar, i.e., the bar with the latest date/time.

Calculation

Looks back the specified number of periods from the specified Bar and returns the Bar number with lowest price within that *Period*.

Example

```
{ Color areas of the chart where the 200 day low has occurred within
the past 20 bars }
var N: float;
var BAR: integer;
for Bar := 200 to BarCount - 1 do
begin
  n := LowestBar( Bar, #Low, 200 );
  if Bar - n <= 20 then
    SetBackgroundColor( Bar, #RedBkg );
end;
```

16.46 MACD

MACD(Bar: integer; Series: integer): float;
MACDSeries(Series: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

MACD returns the Moving Average Convergence Divergence indicator. MACD is a momentum oscillator, yet its primary use is to trade trends. Although it is an oscillator is not used as an over bought or oversold indicator. It appears on the chart

as two lines which oscillates without boundaries. The crossover of the two lines give trading signals similar to a two moving average system.

The two lines are called, MACD Line or fast line and MACD Signal or slow line. The MACD line is displayed as a solid line on the chart, and the MACD signal line is displayed as a dashed line on the chart.

Interpretation

- MACD crossing above zero is considered bullish, and crossing below zero bearish. Secondly, when MACD turns up from below zero it is considered bullish. When it turns down from above zero this is considered bearish.
- Enter a long position and close any short positions when the MACD fast line crosses from below to above the signal line. The further below the zero line the stronger the signal.
- Enter a short position and close any long positions when the MACD fast line crosses from above to below the signal line. The further above the zero line the stronger the signal.
- Divergence between the MACD and the price action is a strong signal when it confirms the crossover signals.
- During trading ranges the MACD will whipsaw, the fast line crosses back and forth across the signal line. Avoid trading or cut your losses very quickly.

Calculation

An *approximated* MACD can be constructed by subtracting the value of a 26 day Exponential Moving Average (EMA) from a 12 period EMA. The shorter EMA is constantly converging toward, and diverging away from, the longer EMA. This causes MACD to oscillate around the zero level.

$$\begin{aligned} \text{MACD line} &= \text{EMA}(12, \text{close}) - \text{EMA}(26, \text{close}), \text{ and} \\ \text{MACD Signal} &= \text{EMA}(9, \text{MACD Line}) \end{aligned}$$

where,

EMA= Exponential Moving Average

MACD line = MACD fast line, displayed as a solid line on the chart

MACD Signal = MACD signal line or slow line, displayed as a dashed line on the chart

The classical MACD calculation, Wealth-Lab's MACD indicator, is based on 2 EMAs with exponents 0.075 and 0.15. A 26 period EMA has an exponent of 0.074074 and the 12 has 0.153846. If you want to use approximate MACD instead of the classical indicator you can use **MACDEX**, a custom indicator that lets you provide 2 periods for EMA.

Note: UseUpdatedEMA affects the calculation of EMA-based indicators such as **MACD**. Choose the default method for calculating the EMA exponent in the *Indicator Calculations* section of the Options dialog.

Example

```
{ This system buys a new position whenever MACD crosses the signal line
from below 0.
  It sells all open positions when MACD crosses below the signal line
from above 0.
  The trading loop starts at Bar 60 in order to give the 26-period EMA
used in the MACD calculation time to stabilize.
}
var MACDPANE, MACDSIGNAL, BAR, P: integer;
```

```

MACDPane := CreatePane( 100, true, true );
PlotSeries( MACDSeries( #Close ), MACDPane, 500, #Histogram );
MACDSignal := EMASeries( MACDSeries( #Close ), 9 );
PlotSeries( MACDSignal, MACDPane, #Black, #Thin );
for Bar := 60 to BarCount - 1 do
begin
  if CrossOver( Bar, MACDSeries( #Close ), MACDSignal ) then
    if MACD( Bar, #Close ) < 0 then
      BuyAtMarket( Bar + 1, '' );
  if CrossUnder( Bar, MACDSeries( #Close ), MACDSignal ) then
    if MACD( Bar, #Close ) > 0 then
      SellAtMarket( Bar + 1, #All, 'MACD' );
end;

```

16.47 MAMA

MAMA(Bar: integer; Series: integer; FastLimit: float; SlowLimit: float): float;
MAMASeries(Series: integer; FastLimit: float; SlowLimit: float): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

MAMA stands for MESA Adaptive Moving Average. It was developed by John Ehlers of Mesa Software, and presented in the September 2001 issue of Stocks & Commodities magazine. MAMA is an adaptive exponential moving average. The EMA's alpha (a) is related to the phase rate of change (the degree to which the phase of the market cycle changes from bar to bar).

In addition to Price Series, MAMA accepts two additional parameters, *FastLimit* and *SlowLimit*. These control the maximum and minimum alpha (a) value that should be applied to the most recent bar of data when calculating MAMA.

You can learn more about the Mesa Adaptive Moving Average at the www.mesasoftware.com web site.

Interpretation

- MAMA is a type of moving average. You can use it in place of any other moving average, and apply the same interpretations, such as price crossovers, crossovers of short and long period averages, etc. MAMA crossovers typically exhibit fewer whipsaws than traditionally moving averages.
- MAMA is also used in conjunction with its complimentary FAMA indicator. Trading signals occur when MAMA crosses over and under FAMA.

Calculation

$$\text{MAMA} = \text{alpha} * \text{Price} + (1 - \text{alpha}) * \text{Previous MAMA value}$$

This is a typical exponential moving average calculation. The difference is that the alpha value changes bar by bar, and is based on the following formula:

$$\text{Alpha} = \text{FastLimit} / \text{DeltaPhase}$$

DeltaPhase is the rate of change of the Hilbert Transform homodyne discriminator. The alpha value is kept within the range of *FastLimit* and *SlowLimit*.

Example

```

var Bar, hMA, hFA: integer;
hMA := MAMASeries( #Close, 0.5, 0.05 );

```

```

hFA := FAMASeries( #Close, 0.5, 0.05 );
PlotSeries( hMA, 0, #Red, #Thin );
PlotSeries( hFA, 0, #Blue, #Thin );
for Bar := 40 to BarCount - 1 do
begin
  if CrossOver( Bar, hMA, hFA ) then
    BuyAtMarket( Bar + 1, '' )
  else if CrossOver( Bar, hFA, hMA ) then
    SellAtMarket( Bar + 1, LastPosition, '' );
end;

```

16.48 Median

Median(Bar: integer; Series: integer; Period: integer): float;
 MedianSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Median returns the Median value of a *Series* based on the specified *Period*. Median sorts the values and returns the value occupying the "middle" slot of the group. When there is an odd number of values, the median is simply the middle value. For example, the median of 2, 4, and 7 is 4. When there is an even number of values, the median is the average of the two middle numbers. Thus, the median of the numbers 2, 4, 7, 12 is $(4+7)/2 = 5.5$.

Example

```

var Bar, hMedFast, hMedSlow: integer;
hMedFast := MedianSeries( #Close, 13 );
hMedSlow := MedianSeries( #Close, 25 );
PlotSeries( hMedFast, 0, 520, #Thick );
DrawLabel( 'Median(Close,13)', 0 );
PlotSeries( hMedSlow, 0, 200, #Thick );
DrawLabel( 'Median(Close,25)', 0 );
for Bar := 25 to BarCount - 1 do
begin
  if CrossOver( Bar, hMedFast, hMedSlow ) then
    BuyAtMarket( Bar + 1, '' )
  else if CrossUnder( Bar, hMedFast, hMedSlow ) then
    SellAtMarket( Bar + 1, LastPosition, '' );
end;

```

16.49 MFI

MFI(Bar: integer; Period: integer): float;
 MFISeries(Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Money Flow Index measures the flow of money into and out of a security over the specified *Period*. Its calculation is similar to that of the Relative Strength Index (RSI), but takes volume into account in its calculation. The indicator is calculated by accumulating positive and negative Money Flow values (see Money Flow indicator), then creating a Money Ratio. The Money Ratio is then normalized into the MFI oscillator form.

Interpretation

- Look for oversold levels below 20 and overbought levels above 80. These normally occur before the underlying price chart forms a top or a bottom. Levels may change depending on market conditions. Ensure that the level lines cut across the highest peaks and the lowest troughs. During strong trends the MFI may remain in overbought or oversold for extended periods.
- If underlying price makes a new high or low that isn't confirmed by the MFI, this divergence can signal a price reversal. MFI divergences from price indicates very strong buy or sell signal.
- The mid point level of 50 will often act as support or resistance if the FMI bounce off the 50 level. Crosses of the 50 level can be used as a buying or selling signal. When MFI cross above then buy, when FMI crosses below then sell.

Calculation

The follow steps are used to calculate Money Flow Index. See **MoneyFlow** Indicator for an excellent example script showing the construction of the MFI.

```
Average Price = #AverageC = ( High + Low + Close ) / 3
Money Flow = Volume x Average Price
```

Money Flow direction: if today's average price is greater than yesterday's, then it is considered positive money flow, otherwise it is negative money flow.

Positive Money Flow = Sum all the Positive Money Flows day over specified periods.
 Negative Money Flow = Sum all the Negative Money Flows day over specified periods.
 Money Ratio = Sum of Positive Money Flow / Sum of Negative Money Flow

```
Money Flow Index (MFI) = 100 - ( 100 / ( 1 + Money Ratio ) )
```

Example

```
{ The trading system below buys a position whenever MFI
  crosses below 20. It sells all open positions as soon
  as MFI crosses above 80. The ChartScript also colors
  MFI bars red and green to show oversold/overbought levels. }
```

```
var MFIPANE, MFISer, BAR, P: integer;
MFISer := MFISeries( 14 );
MFIPane := CreatePane( 100, true, true );
PlotSeries( MFISer, MFIPane, #Black, #Thick );
DrawLabel( 'MFISer = MFI( 14 )', MFIPane );
for Bar := 14 to BarCount - 1 do
begin
  if CrossUnderValue( Bar, MFISer, 20 ) then
    BuyAtMarket( Bar + 1, '' );
  if CrossOverValue( Bar, MFISer, 80 ) then
    for P := 0 to PositionCount - 1 do
      if PositionActive( P ) then
        SellAtMarket( Bar + 1, P, 'MFI' );
  if MFI( Bar, 14 ) < 20 then
    SetSeriesBarColor( Bar, MFISer, #Red );
  if MFI( Bar, 14 ) > 80 then
    SetSeriesBarColor( Bar, MFISer, #Green );
end;
```


16.50 Momentum

Momentum(Bar: integer; Series: integer; Period: integer): float;
MomentumSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Momentum is the difference between current price and the price a specified number of bars ago, *Period*. The momentum indicators shows the speed at which price changes from one period to another. It give a excellent indication of the market participants commitment to the current trend. When the momentum begins to slow or turn, it indicates diminishing commitment and a loss of momentum. This indicator is a leading or coincidental indicator. A momentum value above zero indicates that prices are moving up, and below zero moving down.

The momentum indicator has overbought and oversold zones. These zones are defined by lines that are placed so the Momentum indicator spends about 5% of its time within the zones. The lines should be adjust according to market conditions.

Interpretation

- In ranging markets, go long when the indicator falls below the oversold line then rises back above the oversold line.
- In ranging markets, go short when indicator rises above the overbought line the falls back below the overbought line.
- In ranging markets, go long on bullish divergences, if the indicator's first trough is in the oversold zone.
- In ranging markets, go short on bearish divergences, if the indicator's first peak is in the overbought zone.
- An uptrend can be confirmed using a trend following indicator. Go long when the momentum indicator turns up from below the center line. Exit using the trend following indicator. Divergences of the momentum and price in during the trend can be misleading.
- A downtrend can be confirmed using a trend following indicator. Go short when the indicator turns down from above the center line. Exit using the trend following indicator. Divergences of the momentum and price in during the trend can be misleading.

Calculation

Momentum = (Price today) - (Price n periods ago)

Typically, the closing Price Series, **#Close**, is used.

Example

```
{ This ChartScript plots absolute momentum, and calculates momentum as
a percentage of current price. }
var MOMPANE, MOMPCTPANE, MOMPCT: integer;
MomPane := CreatePane( 100, true, true );
MomPctPane := CreatePane( 100, true, true );
PlotSeries( MomentumSeries( #Close, 30 ), MomPane, 202, #ThickHist );
DrawLabel( 'Standard Momentum', MomPane );
MomPct := DivideSeries( MomentumSeries( #Close, 30 ), #Close );
MomPct := MultiplySeriesValue( MomPct, 100 );
PlotSeries( MomPct, MomPctPane, 022, #ThickHist );
DrawLabel( 'Percentage Momentum', MomPctPane );
```

16.51 MomentumPct

MomentumPct(Bar: integer; Series: integer; Period: integer): float;
 MomentumPctSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

MomentumPct is the current price divided by the price of a previous *Period*. Further, the quotient is multiplied by 100. The result is an indicator that oscillates around 100. Values less than 100 indicate negative momentum, or decreasing price, and vice versa.

Interpretation

- **MomentumPct** can be interpreted in a similar way as the standard **Momentum** indicator. However, **MomentumPct** has the additional advantage of indicating the *amount* of commitment to the current trend in a consistent manner over a broad range of prices. For example, assume that ABC is priced at 60 and XYZ is quoted at 20. After X *Periods*, ABC is now 62 and XYZ is 22. Though **Momentum** is the same (2.0) for both issues, **MomentumPct** is 103.33 and 110.0, respectively, indicating that the price movement is more significant for the lower-priced security, i.e., 10% vs. 3.33%.
- Subtract a constant 100 from MomentumPct to yield the absolute percentage change over the specified *Period* to yield the same result as the **ROC** indicator.

Calculation

$$\text{MomentumPct} = 100 * (\text{Current Price}) / (\text{Price } n \text{ periods ago})$$

Example

```
{ Duplicate the MomentumPctSeries calculation }
var Bar, Period, hTmp, hMomPct, MomPctPane: integer;
Period := 20;
MomPctPane := CreatePane( 100, true, true );
hTmp := OffsetSeries( #Close, -Period );
hMomPct := MultiplySeriesValue( DivideSeries( #Close, hTmp ), 100 );
PlotSeries( hMomPct, MomPctPane, #Red, #Thick );
PlotSeries( MomentumPctSeries( #Close, Period ), MomPctPane, #Black,
#Thin );
```

16.52 MoneyFlow

MoneyFlow(Bar: integer): float;
 MoneyFlowSeries: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Money Flow returns the average price multiplied by volume. Money Flow is the core component of the Money Flow Index (MFI) indicator. This is not really an indicator, but a mathematical function used to construct other indicators.

Interpretation

See the Money Flow Index (MFI) indicator and the example script application below.

Calculation

Money Flow is the average price multiplied by Volume.

$$\text{Average Price} = \#AverageC = (\text{High} + \text{Low} + \text{Close}) / 3$$

$$\text{Money Flow} = \text{Volume} \times \text{Average Price}$$

Example

```
{ The example below duplicates the calculation of the MFI }
var TODAY, YESTERDAY, X: float;
var MFPOSITIVE, MFNEGATIVE, MYMFI, BAR, MFPOSSUM, MFNEGSUM, MONEyratio,
MFPANE, MFIPANE: integer;
MFPositive := CreateSeries;
MFNegative := CreateSeries;
MyMFI := CreateSeries;

for Bar := 1 to BarCount - 1 do
begin
  today := PriceAverageC( Bar );
  yesterday := PriceAverageC( Bar - 1 );
  if today > yesterday then
    SetSeriesValue( Bar, MFPositive, MoneyFlow( Bar ) )
  else if today < yesterday then
    SetSeriesValue( Bar, MFNegative, MoneyFlow( Bar ) );
end;

MFPosSum := SumSeries( MFPositive, 14 );
MFNegSum := SumSeries( MFNegative, 14 );
MoneyRatio := DivideSeries( MFPosSum, MFNegSum );

for Bar := 14 to BarCount - 1 do
begin
  x := 100 - ( 100 / ( 1 + GetSeriesValue( Bar, MoneyRatio ) ) );
  SetSeriesValue( Bar, MyMFI, x );
end;

MFPANE := CreatePane( 100, true, true );
PlotSeries( MyMFI, MFPANE, #Navy, #Thick );
MFIPANE := CreatePane( 100, true, true );
PlotSeries( MFISeries( 14 ), MFIPANE, 950, #Thick );
DrawLabel( 'MFI( 14 )', MFIPANE );
```

16.53 NVI

NVI(Bar: integer): float;
 NVISeries: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Negative Volume Index was created by Norman Fosback, and its purpose is to expose where "smart money" investment action is occurring. The assumption is that smart money, mostly floor traders, will produce moves in price with less volume than the rest of the crowd.

Interpretation

Fosback compared the NVI with its one year (255 bar) moving average. When NVI is above the moving average, he calculated that there is a 96% chance that a bull market is in progress, and when it is below the average a 53% chance of a bear market.

Example

```

var Bar: integer;
var NVIPane, PVIPane: integer;
NVIPane := CreatePane( 75, true, true );
PVIPane := CreatePane( 75, true, true );
var NVI1, PVI1, SMA1, SMA2: integer;
NVI1 := NVISeries;
PVI1 := PVISeries;
SMA1 := SMASeries( PVI1, 255 );
SMA2 := SMASeries( NVI1, 255 );
PlotSeriesLabel( NVI1, NVIPane, 900, #Thick, 'NVI1=NVI()' );
PlotSeriesLabel( PVI1, PVIPane, 050, #Thick, 'PVI1=PVI()' );
PlotSeriesLabel( SMA1, PVIPane, 020, #Thin, 'SMA1=SMA(PVI1,255)' );
PlotSeriesLabel( SMA2, NVIPane, 200, #Thin, 'SMA2=SMA(NVI1,255)' );
Bar := BarCount - 1;
if PVI( Bar ) > @SMA1[Bar] then
  DrawLabel( 'PVI says 79% chance Bull Market is in progress', 0 )
else
  DrawLabel( 'PVI says 67% chance a Bear Market is in progress', 0 );
if NVI( Bar ) > @SMA2[Bar] then
  DrawLabel( 'NVI says 96% chance Bull Market is in progress', 0 )
else
  DrawLabel( 'NVI says 53% chance a Bear Market is in progress', 0 );

```

16.54 OBV

OBV(Bar: integer): float;
 OBVSeries: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

On Balance Volume developed by Joseph Granville and described in his "New Key to Stock Market Profits", uses volume to gauge the strength of a market. If prices close up, the current bar's volume is added to OBV, and if prices close down, it is subtracted. The result is an indicator that depicts the flow of volume into and out of a security. It either confirms the quality of the current trend or warn of an impending reversals.

You can often spot divergences between price action and the OBV indicator. For example, if prices make a new high but the move is not accompanied by sufficient volume, OBV will fail to make a new high. Such divergences can be a sign that a trend is nearing completion.

Interpretation

The actual value of the OBV is unimportant, concentrate on its direction.

- When both price and OBV are making higher peaks and higher troughs, the up trend is likely to continue.
- When both price and OBV are making lower peaks and lower troughs, the down trend is likely to continue.
- When price continues to make higher peaks and OBV fails to make higher peak, the up trend is likely to stall or fail.
- When price continues to make lower troughs and OBV fails to make lower troughs, the down trend is likely to stall or fail.
- If during a trading range, the OBV is rising then accumulation may be taking place

and is a warning of an upward break out.

- If during a trading range, the OBV is falling then distribution may be taking place and is a warning of an downward break out.

Calculation

On Balance Volume is calculated as follows:

" . . . the total daily volume is added to a cumulative total whenever the price of a stock closes higher than the day before and it is subtracted whenever the price of the stock closes lower than the day before. On days when the stock closes unchanged in price, the running cumulative volume remains unchanged." (Granville, p. 144)

Example

```
{ This simple systems buys and sells based
  on a moving average crossover of OBV }
var OBVPANE, OBV1, OBV2, BAR: integer;
OBVPane := CreatePane( 80, false, true );
PlotSeries( OBVSeries, OBVPane, 700, #Thick );
OBV1 := EMASeries( OBVSeries, 24 );
OBV2 := EMASeries( OBVSeries, 48 );
PlotSeries( OBV1, OBVPane, #Black, #Dotted );
PlotSeries( OBV2, OBVPane, #Red, #Dotted );
for Bar := 48 to BarCount - 1 do
begin
  if CrossOver( Bar, OBV1, OBV2 ) then
    BuyAtMarket( Bar + 1, '' )
  else if CrossUnder( Bar, OBV1, OBV2 ) then
    SellAtMarket( Bar + 1, LastPosition, 'OBV' );
end;
```

16.55 Parabolic

```
Parabolic( Bar: integer; AccelUp: float; AccelDown: float; AccelMax: float ): float;
ParabolicSeries( AccelUp: float; AccelDown: float; AccelMax: float ): integer;
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Welles Wilder's Parabolic SAR is actually a type of trailing stop-based system, but it's often used as an indicator. The SAR (Stop And Reverse) uses a trailing stop level that follows prices as they move up or down. The stop level increases speed based on an "Acceleration Factor". When plotted on the chart, this stop level resembles a parabolic curve, thus the indicator's name. The Parabolic function accepts 3 parameters. The first two control the Acceleration during up and down moves, respectively. The last parameter determines the maximum Acceleration.

The Parabolic assumes that you are trading a trend and therefore expects price to change over time. If you are long the Parabolic SAR will move the stop up every period, regardless of whether the price has moved. It moves down if you are short.

Interpretation

- The Parabolic SAR trading system uses the Parabolic level as a Stop and Reverse point. This stop is calculated for the next period. When the stop is hit, this signals to close the trade and take a new trade in the opposite direction. The system is typically always in the market.
- When price movement trades in a narrow trading range, the Parabolic SAR will whipsaw. The Parabolic is trend following indicator, it is useless in the absence of a trend. Use another indicator, such as ADXR, to determine trend strength.
- The Parabolic excels in fast moving trends that accelerate as they progress. The stops are also calculated to accelerate, hence you need to have the correct "Acceleration Factor" to match the market you are trading. Up and down accelerations parameters maybe different.
- The indicator is usually shown as a series of dots above or below the price bars. The dots are the stop levels. You should be short when the stops are above the bars and long when the stops are below the bars. When a stop is hit then trade in opposite direction.

Calculation

$SAR_t = SAR_c + AF * (EP - SAR_c)$, where

SAR_t = the stop for the next bar
 SAR_c = the stop for the current bar
 AF = Acceleration Factor
 EP = Extreme Point for current trade

The AF used by Wilder is 0.02. This means move the stop 2 percent of distance between EP and the original stop. Each time the EP changes, the AF increases by 0.02 up to the maximum acceleration, 0.2 in Wilders' case. Practical values are: AF range 0.01 to 0.025, and AFmax range of 0.1 to 0.25.

If long then EP is the highest high since going long, if short then EP is the lowest low since going short.

Example

```
var Bar: integer;
var x: float;
for Bar := 20 to BarCount - 1 do
begin
  x := Parabolic( Bar, 0.02, 0.02, 0.2 );
  if not LastPositionActive then
  begin
    if PriceLow( Bar ) < Parabolic( Bar, 0.02, 0.02, 0.2 ) then
      BuyAtStop( Bar + 1, x, '' )
    else
      ShortAtStop( Bar + 1, x, '' );
  end
  else
  begin
    if PositionLong( LastPosition ) then
    begin
      SellAtStop( Bar + 1, x, LastPosition, '' );
      ShortAtStop( Bar + 1, x, '' );
    end
    else
    begin
      CoverAtStop( Bar + 1, x, LastPosition, '' );
    end
  end
end
```

```
        BuyAtStop( Bar + 1, x, ' ' );
    end;
end;
end;
PlotSeries( ParabolicSeries( 0.02, 0.02, 0.2 ), 0, 905, #Dots );
DrawLabel( 'Parabolic( 0.02, 0.02, 0.2 )', 0 );
```

16.56 Peak

Peak(Bar: integer; Series: integer; Reversal: float): float;
PeakSeries(Series: integer; Reversal: float): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the value of the last Peak that was identified for the specified Price *Series* as of the specified *Bar*. The *Reversal* parameter determines how much of a percentage (default) or point decline is required to trigger a new Peak. It typically requires a few bars of downward price movement to reach the Reversal level and qualify a new Peak. The Peak function never "looks ahead" in time, but always returns the Peak value as it would have been determined as of the specified bar. For this reason, the return value of the Peak function will lag, and report peaks a few bars later than they actually occurred in hindsight. This is intentional, and allows peak/trough detection to be used when back-testing trading systems.

Interpretation

- Peaks/Troughs of highs and lows are often used as support and resistance levels. These points have historical significance because they have proven to be important levels of price reversal.
- Peaks and troughs are a convenient way of detecting chart patterns. For example, one aspect of a Head & Shoulders Top is a series of 3 Peaks, the second higher than the outer two.

Remarks

- To base reversals on point/absolute movement, pass the **#AsPoint** constant to the **SetPeakTroughMode** function in your script.
- Calculating peaks/troughs based on *percentage* moves is not allowed on data series that contains negative or zero values. For these data series you must use **SetPeakTroughMode** to base the reversal amount on a point value.

Calculation

Peaks are detected by looking for a percentage (default) or point reversal in the Price Series greater than or equal to the reversal amount specified in the Reversal parameter. For example, if you specify a reversal value of 10, and prices make a new high of \$100, a peak will be triggered at that bar as soon as prices move down to \$90 (provided they do not continue above \$100). The move down to \$90 may take several bars. During these bars the Peak function will not return \$100, but will instead return the value of the previous peak. This is because you would not have known that that \$100 was an actual peak yet because the reversal level has not been met. The new Peak value of \$100 will be returned only after prices have reached the \$90 level, and the reversal level is reached.

Example

```
{ Draw the level of 7% Peaks on the chart }
var PS: integer;
PS := PeakSeries( #High, 7 );
PlotSeries( PS, 0, #Red, #Dots );
```

16.57 PeakBar

```
PeakBar( Bar: integer; Series: integer; Reversal: float ): integer;
PeakBarSeries( Series: integer; Reversal: float ): integer;
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the bar number at which the last Peak that was identified for the specified Price *Series* as of the specified *Bar*. The *Reversal* parameter determines how much of a percentage (default) decline is required to trigger a new Peak. It typically requires a few bars of downward price movement to reach the Reversal level and qualify a new Peak. The Peak function never "looks ahead" in time, but always returns the Peak value as it would have been determined as of the specified bar. For this reason, the return value of the Peak function will lag, and report peaks a few bars later than they actually occurred in hindsight. This is intentional, and allows peak/trough detection to be used when back-testing trading systems.

Interpretation

- Peaks/Troughs of highs and lows are often used as support and resistance levels. These points have historical significance because they have proven to be important levels of price reversal.
- Peaks and troughs are a convenient way of detecting chart patterns. For example, one aspect of a Head & Shoulders Top is a series of 3 Peaks, the second higher than the outer two.
- PeakBar is particularly useful with working with chart patterns. You can store the bar number of the most recent peak, then use this as an anchor bar to retrieve the bar number for the previous peak, and so on.

Remarks

- **PeakBar** returns -1 if a peak has not yet been detected at the beginning of the chart.
- To base reversals on point/absolute movement, pass the **#AsPoint** constant to the **SetPeakTroughMode** function in your script.
- Calculating peaks/troughs based on *percentage* moves is not allowed on data series that contains negative or zero values. For these data series you must use **SetPeakTroughMode** to base the reversal amount on a point value.

Calculation

(See [Peak](#)^[220])

Example

```
{ Draw a trendline from the 2 most recent 10% Peaks }
var p1, p2, Bar: integer;
var Detected2Peaks: boolean = false;

Bar := BarCount - 1;
```



```

p1 := PeakBar( Bar, #Close, 10 );
if p1 > -1 then
begin
  p2 := PeakBar( p1, #Close, 10 );
  if p2 > -1 then
  begin
    DrawLine( p1, PriceClose( p1 ), p2, PriceClose( p2 ), 0, #Red,
#Thick );
    Detected2Peaks := true;
  end;
end;
if not Detected2Peaks then
  DrawText( '2 peaks not detected. Try another symbol or load more
data.', 0, 5, 50, #Red, 10 );

```

16.58 PeakNum

PeakNum(Bar: integer; Series: integer; Number: integer; Reversal: float): float;
PeakNumSeries(Series: integer; Number: integer; Reversal: float): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the value of the "Nth" most recent Peak that was identified for the specified Price *Series* as of the specified *Bar*. The *Reversal* parameter determines how much of a percentage (default) decline is required to trigger a new Peak. It typically requires a few bars of downward price movement to reach the Reversal level and qualify a new Peak. The Peak function never "looks ahead" in time, but always returns the Peak value as it would have been determined as of the specified bar. For this reason, the return value of the Peak function will lag, and report peaks a few bars later than they actually occurred in hindsight. This is intentional, and allows peak/trough detection to be used when back-testing trading systems.

Use the *Number* parameter to specify which Peak to identify. To obtain the most recent Peak, pass 0 (although this is the same as using the Peak function). *Number* = 1 returns the previous Peak, 2 returns the second most previous, etc.

Interpretation

- Peaks/Troughs of highs and lows are often used as support and resistance levels. These points have historical significance because they have proven to be important levels of price reversal.
- Peaks and troughs are a convenient way of detecting chart patterns. For example, one aspect of a Head & Shoulders Top is a series of 3 Peaks, the second higher than the outer two.

Remarks

- To base reversals on point/absolute movement, pass the **#AsPoint** constant to the **SetPeakTroughMode** function in your script.
- Calculating peaks/troughs based on *percentage* moves is not allowed on data series that contains negative or zero values. For these data series you must use **SetPeakTroughMode** to base the reversal amount on a point value.

Calculation

Peaks are detected by looking for a percentage (default) or point reversal in the Price Series greater than or equal to the reversal amount specified in the Reversal parameter. For example, if you specify a reversal value of 10, and prices make a new

high of \$100, a peak will be triggered at that bar as soon as prices move down to \$90 (provided they do not continue above \$100). The move down to \$90 may take several bars. During these bars the Peak function will not return \$100, but will instead return the value of the previous peak. This is because you would not have known that that \$100 was an actual peak yet because the reversal level has not been met. The new Peak value of \$100 will be returned only after prices have reached the \$90 level, and the reversal level is reached.

Example

```
{ Flags bars as red when a potential Head & Shoulders top is forming.
Note, this script does not check for penetration of the neckline. }
```

```
var P1, P2, P3, LASTHEAD: float;
var BAR, pb1, pb2, pb3: integer;
for Bar := 120 to BarCount - 1 do
begin
  p1 := Peak( Bar, #High, 7 );
  p2 := PeakNum( Bar, #High, 1, 7 );
  p3 := PeakNum( Bar, #High, 2, 7 );
  if ROC( Bar, #Close, 120 ) > 20 then
    if p1 < p2 then
      if p2 > p3 then
        begin
          if LastHead <> p2 then
            begin
              LastHead := p2;
              pb1 := PeakBar( Bar, #High, 7 );
              pb2 := PeakBar( pb1, #High, 7 );
              pb3 := PeakBar( pb2, #High, 7 );
              AnnotateBar( 'S1', pb3, true, #Black, 8 );
              AnnotateBar( 'H', pb2, true, #Black, 8 );
              AnnotateBar( 'S2', pb1, true, #Black, 8 );
            end;
          if PriceHigh( Bar ) < LastHead then
            SetBarColor( Bar, #Red );
        end;
      end;
    end;
end;
```

16.59 PVI

```
PVI( Bar: integer ): float;
PVISeries: integer;
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Positive Volume Index was created by Norman Fosback, and its purpose is to expose where "smart money" investment action is occurring. The assumption is that smart money, mostly floor traders, will produce moves in price with less volume than the rest of the crowd.

Interpretation

Fosback compared the PVI with its one year (255 bar) moving average. When PVI is above the moving average, he calculated that there is a 79% chance that there is a bull market in progress, and when it is below the average a 67% chance of a bear market.

Example

```
var Bar: integer;
```

```

var NVIPane, PVIPane: integer;
NVIPane := CreatePane( 75, true, true );
PVIPane := CreatePane( 75, true, true );
var NVI1, PVI1, SMA1, SMA2: integer;
NVI1 := NVISeries;
PVI1 := PVISeries;
SMA1 := SMASeries( PVI1, 255 );
SMA2 := SMASeries( NVI1, 255 );
PlotSeriesLabel( NVI1, NVIPane, 900, #Thick, 'NVI1=NVI()' );
PlotSeriesLabel( PVI1, PVIPane, 050, #Thick, 'PVI1=PVI()' );
PlotSeriesLabel( SMA1, PVIPane, 020, #Thin, 'SMA1=SMA(PVI1,255)' );
PlotSeriesLabel( SMA2, NVIPane, 200, #Thin, 'SMA2=SMA(NVI1,255)' );
Bar := BarCount - 1;
if PVI( Bar ) > @SMA1[Bar] then
  DrawLabel( 'PVI says 79% chance Bull Market is in progress', 0 )
else
  DrawLabel( 'PVI says 67% chance a Bear Market is in progress', 0 );
if NVI( Bar ) > @SMA2[Bar] then
  DrawLabel( 'NVI says 96% chance Bull Market is in progress', 0 )
else
  DrawLabel( 'NVI says 53% chance a Bear Market is in progress', 0 );

```

16.60 QStick

```

QStick( Bar: integer; Period: integer ): float;
QStickSeries( Period: integer ): integer;

```

ChartScripts SimuScripts PerfScripts CMScripts

Description

QStick provides a way to quantify candlestick values. The **QStick** indicator is calculated by taking a moving average of the difference between open and closing prices.

Interpretation

When **QStick** crosses above zero, this is considered bullish, and below zero bearish. You can also look for extreme **QStick** levels to determine overbought and oversold levels, or look for divergences between **QStick** and price to signal trend reversals.

Example

```

{ See how good the QStick zero line entry rule really is }
var QSTICKPANE, BAR: integer;
QStickPane := CreatePane( 100, true, true );
PlotSeries( QStickSeries( 24 ), QStickPane, 050, #Thick );
DrawLabel( 'QStick( 24 )', QStickPane );
InstallProfitTarget( 10 );
InstallStopLoss( 20 );
for Bar := 24 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  if CrossOverValue( Bar, QStickSeries( 24 ), 0 ) then
    BuyAtMarket( Bar + 1, '' );
  if CrossUnderValue( Bar, QStickSeries( 24 ), 0 ) then
    ShortAtMarket( Bar + 1, '' );
end;

```

16.61 RelSlope

RelSlope(Bar: integer; Series: integer; Period: integer; Smooth: integer): float;
 RelSlopeSeries(Series: integer; Period: integer; Smooth: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

RelSlope stands for the "Relative Slope" Indicator, created by Dimitris Tsokakis. RelSlope takes 3 parameters. The *Series* parameter specifies what you wish to apply Relative Slope to, average price weighted with closing price is recommended (**#AverageC**).

Note: *Series* should be a Price Series that contains positive values only. Calculating RelSlope on a Price Series that contains negative values is meaningless.

The *Period* parameter determines the period of an initial **EMA** that is taken of the *Series*. A *Period* of 10 is recommended. The final parameter, *Smooth*, determines a final smoothing of the indicator, and a value of 3 is recommended.

Interpretation

As an independent indicator, RelSlope is a fast trend follower and its divergences often anticipate big price movements.

Note: **UseUpdatedEMA** affects the calculation of EMA-based indicators such as **RelSlope**. Choose the default method for calculating the EMA exponent in the *Indicator Calculations* section of the Options dialog.

Calculation

Example

```
{ This ChartScript demonstrates the calculation of RelSlope }
var Period, Smooth, RSPane: integer;
var K, KPLUS, KMINUS, S1, MyRelSlopeSeries: integer;
UseUpdatedEMA( true );
Period := 10;
Smooth := 3;

K := EMASeries( #Close, Period );
KPlus := AddSeries( K, OffsetSeries( K, -1 ) );
KMinus := SubtractSeries( K, OffsetSeries( K, -1 ) );
S1 := MultiplySeriesValue( DivideSeries( KMinus, KPlus ), 2 );
MyRelSlopeSeries := MultiplySeriesValue( EMASeries( S1, Smooth ), 1000
);

RSPane := CreatePane( 75, true, true );
PlotSeriesLabel( MyRelSlopeSeries, RSPane, #Blue, #Histogram,
'RelSlope(calculated)' );
PlotSeriesLabel( RelSlopeSeries( #Close, Period, Smooth ), RSPane,
#Red, #Thin, 'RelSlope' );
```

16.62 ROC

ROC(Bar: integer; Series: integer; Period: integer): float;
 ROCSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Rate of Change (ROC) indicator provides a percentage that the security's price has changed over the specified *Period*. The Rate of Change shows the speed at which price changes from one period to another. Sometimes this is referred to as momentum. It gives an excellent indication of the market participants' commitment to the current trend. When the ROC begins to reverse or turn, it indicates diminishing commitment and a loss of momentum. ROC is a leading or coincidental indicator.

Like other momentum indicators, ROC has overbought and oversold zones. These zones are defined by lines that are placed so that ROC spends about 5% of its time within the zones. The lines should be adjusted according to market conditions.

Interpretation

- In ranging markets, go long after ROC falls below the oversold line then rises back above it.
- In ranging markets, go short after ROC rises above the overbought line then falls back below it.
- In ranging markets, go long on bullish divergences if ROC's first trough is in the oversold zone.
- In ranging markets, go short on bearish divergences if ROC's first peak is in the overbought zone.
- In an up trend confirmed by a trend-following indicator, go long when ROC turns up from below the center line. Exit using the trend following indicator. Divergences of ROC and price during a trend can be misleading.
- In a down trend, confirmed by a trend-following indicator, go short when the ROC turns down from above the center line. Exit using the trend following indicator. Divergences of ROC and price during trend can be misleading.

Calculation

ROC is the percentage change between the current price with respect to an earlier price. Typically, the closing Price Series (#Close) is used.

$$\text{ROC}(\text{Bar}) = 100 * ((\text{Price}(\text{Bar}) / \text{Price}(\text{Bar} - \text{Period})) - 1),$$

where Bar is the current Bar. For example, if the current price is 77 and the previous price were 70, $\text{ROC} = 100 * ((77 / 70) - 1) = 10.0$, which is the percentage change from 70.

Example

```
{ This system is based on a smoothed Rate of Change. Entry occurs when
smoothed ROC rises above zero. The long Position is closed when the
smoothed ROC turns down. }
var ROCPANE, SMAROC, BAR: integer;
ROCPane := CreatePane( 75, true, true );
PlotSeries( ROCSeries( #Close, 40 ), ROCPane, 005, #ThickHist );
SMARoc := SMASeries( ROCSeries( #Close, 40 ), 14 );
PlotSeries( SMARoc, ROCPane, #Black, #Dotted );
for Bar := 54 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    if CrossOverValue( Bar, SMARoc, 0 ) then
      BuyAtMarket( Bar + 1, '' );
    end
  else
  begin
```

```

    if TurnDown( Bar, SMARoc ) then
        SellAtMarket( Bar + 1, LastPosition, '' );
    end;
end;

```

16.63 RSI

RSI(Bar: integer; Series: integer; Period: integer): float;
 RSIseries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

The RSI function returns the Relative Strength Index indicator. RSI is one of the classic momentum indicators and was developed by Wells Wilder. RSI measures a market's internal strength by dividing the average of the sum of the up day closing prices by the the average of the sum of the down day closing prices over a specific period of time. It returns a value within the range of 0 to 100. The RSI is a leading or a coincidental indicator. Popular averaging periods for the RSI are 9, 14 and 25. Wilder used 14 periods. Use the *Period* that works best for you. The indicator becomes more volatile and amplitude widens with fewer periods used.

Interpretation

- The classic way to interpret RSI is to look for oversold levels below 30 and overbought levels above 70. These normally occur before the underlying price chart forms a top or a bottom. Note you should change the levels depending on market conditions. Ensure the level lines cut across the highest peaks and the lowest troughs. During strong trends the RSI may remain in overbought or oversold for extended periods.
- RSI also often forms chart patterns which may not show on the underlying price chart, such as double tops and bottoms and trendlines. Also look for support or resistance on the RSI.
- If underlying prices make a new high or low that isn't confirmed by the RSI this divergence can signal a price reversal. RSI divergences from price indicates very strong buy or sell signal.
- Swing Failures. If the RSI makes a lower high followed buy a downside move below a previous low, then a Top Swing Failure has occurred. If the RSI makes a higher low followed buy a upside move above a previous high, then a Bottom Swing Failure has occurred.
- The mid point level of 50 will often act as support or resistance if the RSI bounce off the 50 level. Crosses of the 50 level can be used as a buying or selling signal. When RSI cross above then buy, when RSI crosses below then sell.
- RSI can be use to find dips in strong trends. Use trend indicator to determine a strong up trend then if the RSI is below 50, you have a dip in the up trend. In strong down trends use RSI above 50 to detect small rallies. Buy the dip and sell the small rally.

Remarks

- As a rule of thumb, allow **RSI** to stabilize for 2.5 to 3 times the specified *Period*. For example, start the trading loop at Bar Number 42 for a 14-period RSI.

Calculation

$$RSI = 100 - (100 / (1 + RS))$$

where,

RSI relative strength index

$$RS = (\text{average of } n \text{ bars' up closes}) / (\text{average of } n \text{ bars' down closes})$$

n = number of bars or period, typically 14

Note in calculating the RS values for the total of closes up, add all price changes where the close is greater than previous close. For closes down, add all price changes where the close is less than previous close.

Finally, the RSI formula may be found in some technical references as the following equivalent expression:

$$RSI = 100 * \text{UpDaysAvg} / (\text{UpDaysAvg} + \text{DownDaysAvg})$$

Example

```
{ This script colors each bar based on the RSI oversold/overbought level }
var X: float;
var RSIPANE, BAR, COL: integer;
RSIPane := CreatePane( 75, true, true );
SetPaneMinMax( RSIPane, 0, 100 );
PlotSeries( RSISeries( #Close, 14 ), RSIPane, 005, #Thin );
DrawLabel( 'RSI( Close, 14 )', RSIPane );
for Bar := 42 to BarCount - 1 do
begin
  x := RSI( Bar, #Close, 14 );
  if x > 50 then
  begin
    x := x - 50;
    x := x * 2;
    x := x / 9;
    col := Trunc( x ) * 100;
  end
  else
  begin
    x := 50 - x;
    x := x * 2;
    x := x / 9;
    col := Trunc( x ) * 10;
  end;
  SetBarColor( Bar, col );
end;
```

16.64 RSquared

RSquared(Bar: integer; Series: integer; Period: integer): float;

RSquaredSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

RSquared is the Correlation Coefficient squared from Linear Regression. It is used to determine how much of the price action over the specified period can be explained by the regression line, and how much should be attributed to random noise. **RSquared** ranges from 0 to 1.

RSquared is a statistical indicator. Other indicators in the same class are **LinearReg**, **LinearRegSlope**, **StdErr**, and **StdDev**.

Interpretation

- The closer **RSquared** is to one, the closer prices have fitted to the linear regression line. See the table below. During strong trends, **RSquared** will remain above 0.5 for an extended period of time. Use the **RSquared** indicator with **LinearRegSlope** to determine if a significant trend is in place.
- Use **RSquared** for confirmation of the trend. When RSI, Stochastics, CCI and other momentum indicators are in overbought or oversold regions, look for **RSquared** to show that no statistical trend is in place before taking a contrary trading position.
- If trading a trend-following system, such as moving average crossover, you can use **RSquared** to confirm that the trend is statistically significant.

Table

The following table show the RSquared values for a given number of periods for a statistically significant trend to be in place. A 95% confidence means that 95% of the prices can be explained by Linear Regression and 5% by unexplained random noise.

Number of periods	RSquared values for 95% confidence
5	0.77
10	0.40
14	0.27
20	0.20
25	0.16
30	0.13
50	0.08
60	0.06
120	0.03

Calculation

RSquared is a rather complex statistical calculation. It uses the least square method to fit a trendline to the data by minimizing the distance between the price and the Linear Regression trendline and returns a percentage of price movement that is explained by the regression line.

Example

```
{ Plot RSquared in order to examine how prices react when they reach
different levels. }
var RSquaredPane: integer;
RSquaredPane := CreatePane( 75, true, true );
PlotSeries( RSquaredSeries( #Close, 30 ), RSquaredPane, 905, #Thin );
DrawLabel( 'RSquared( Close, 30 )', RSquaredPane );
var LinRegSlopePane: integer;
PlotSeries( LinearRegSlopeSeries( #Close, 30 ), RSquaredPane, 509,
#Thin );
DrawLabel( 'LinearRegSlope( Close, 30 )', RSquaredPane );
```

16.65 RVI

```
RVI( Bar: integer; Period: integer ): float;
RVISeries( Period: integer ): integer;
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

RVI returns the Relative Vigor Index, and indicator created by John Ehlers of Mesa Software (<http://www.mesasoftware.com>). **RVI** measures the average difference between closing and opening price, normalized to the average daily trading range. It applies a normalization filter to smooth the index with minimal lag.

Interpretation

RVI reaches extreme high and low levels near the peaks of uptrends and downtrends. You can trigger signals based on these extreme levels, or wait until **RVI** crosses above or below a signal line.

The RVI indicator accepts a parameter that determines the *Period* to use in its calculation. You can create a dynamic **RVI** that is based on half of the dominant cycle period as described by Ehlers. Below we create a custom Price Series and populate with the **RVI** values based on half of the cycle period as determined by the **HTPeriod** indicator.

Example

```

var RVIPANE, DYNRVI, BAR, N, P: integer;
RVIPane := CreatePane( 100, true, true );
DynRVI := CreateSeries;
for Bar := 40 to BarCount - 1 do
begin
  n := Round( HTPeriod( Bar, #Average ) ) div 2;
  SetSeriesValue( Bar, DynRVI, RVI( Bar, n ) );
end;
PlotSeries( DynRVI, RVIPane, 009, #Thick );
DrawLabel( 'DynRVI', RVIPane );
for Bar := 20 to BarCount - 1 do
begin
  if TurnUp( Bar, DynRVI ) then
    if GetSeriesValue( Bar - 1, DynRVI ) < -0.35 then
      BuyAtMarket( Bar + 1, '' );
  if CrossOverValue( Bar, DynRVI, 0.35 ) then
    for P := 0 to PositionCount - 1 do
      if PositionActive( P ) then
        SellAtMarket( Bar + 1, P, '' );
end;
var HTPeriodPane: integer;
HTPeriodPane := CreatePane( 100, true, true );
PlotSeries( HTPeriodSeries( #Average ), HTPeriodPane, 055, #Thick );
DrawLabel( 'HTPeriod( Average )', HTPeriodPane );

```

16.66 SMA

SMA(Bar: integer; Series: integer; Period: integer): float;
 SMASeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

SMA returns the **Simple Moving Average** indicator. Moving averages are one of the core indicators in technical analysis, and there are a variety of different versions. SMA is the easiest moving average to construct. It is simply the average price over the specified *Period*. The average is called "Moving" because it is plotted on the chart bar by bar, forming a line that moves along the chart as the average value changes.

Interpretation

- SMAs are often used to determine **Trend Direction**. If the SMA is moving up, the trend is up, moving down and the trend is down. A 200 bar SMA is common proxy for the long term trend. 60 bar SMAs are typically used to gauge the intermediate trend. Shorter period SMAs can be used to determine shorter term trends.
- SMAs are commonly used to **smooth** price data and technical indicators. Applying an SMA smoothes out choppy data. The longer the period of the SMA, the smoother the result, but the more lag that is introduced between the SMA and the source.
- **SMA Crossing Price** is often used to trigger trading signals. When prices cross above the SMA go long, when they cross below the SMA go short.
- **SMA Crossing SMA** is another common trading signal. When a short period SMA crosses above a long period SMA, go long. Go short when the short term SMA crosses back below the long term.

Calculation

SMA is simply the mean, or average, of the values in a *Series* over the specified *Period*.

Example

```
{ An SMA Crossover system }
var BAR, hSlow, hFast, SlowPer, FastPer: integer;
SlowPer := 100;
FastPer := 40;
hSlow := SMASeries( #Close, SlowPer );
hFast := SMASeries( #Close, FastPer );
PlotSeries( hSlow, 0, 000, #Thick );
PlotSeries( hFast, 0, 502, #Thick );

for Bar := SlowPer to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    if CrossOver( Bar, hFast, hSlow ) then
      BuyAtMarket( Bar + 1, '' );
    end
  else
  begin
    if CrossUnder( Bar, hFast, hSlow ) then
      SellAtMarket( Bar + 1, LastPosition, '' );
    end;
  end;
end;
```

16.67 StdDev

StdDev(Bar: integer; Series: integer; Period: integer): float;
StdDevSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Standard Deviation is the statistical measure of market volatility. If prices trade in a tight narrow trading range then StdDev will return a low value indicating volatility is low. Conversely if prices swing wildly up and down then StdDev returns a high value indicating volatility is high. What it does is measure how widely prices are dispersed from the average or mean price.

Interpretation

- Standard deviation rises as prices become more volatile. As price action calms, standard deviation heads lower.
- Market tops accompanied by increase volatility over short periods of time, indicate nervous and indecisive traders. Or market tops with decreasing volatility over long time frames, indicate maturing bull markets.
- Market bottoms accompanied by decreased volatility over long periods of time, indicate bored and disinterested traders. Or market bottoms with increasing volatility over relatively sort time periods, indicate panic sell off.

Calculation

You can choose between standard deviation *of a sample* (compatible with Excel STDEV) or *of a population* (compatible with Excel STDEVP) in the *Indicator Calculations* section of the Options dialog. See the User Guide for details.

Example

```
{ Divide Standard Deviation by Average Price to arrive
  at a normalized Volatility indicator }
var MYVOLATILITY, VOLPANE: integer;
MyVolatility := DivideSeries( StdDevSeries( #Close, 30 ),
                             SMASeries( #Close, 30 ) );
VolPane := CreatePane( 100, true, true );
PlotSeries( MyVolatility, VolPane, #Purple, #ThickHist );
```

16.68 StdError

StdError(Bar: integer; Series: integer; Period: integer): float;
StdErrorSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the Standard Error of the estimate for a Linear Regression line of the specified *Period*. Standard Error measures the difference between actual price and the estimated price of the Linear Regression line at every point along the line. The lower the standard error, the closer actual prices have met the estimate. If all the closing prices matched the Linear Regression values for the specified period, then the Standard Error would be Zero.

Interpretation

- The larger the error the less reliable the trend as the price has greater variance around the Linear Regression line, prices are volatile. This can be caused by the changes in the prevailing trend within the specified number of periods.
- The smaller the error then more reliable the trend as the prices are congregating around the Linear Regression Linear line.
- If RSquared and Standard Error are at extreme levels and then they begin to converge then expect a change in the trend.

Calculation

Standard Error is a fairly complex statistical calculation. It uses the least square fit method to fit a trendline to the data by minimizing the distance between the price and the Linear Regression trendline. This is used to find an estimated of the next periods price. The Standard Error indicator returns the statistical difference between the estimate and the actual price.

Example

```

{ Display the most recent Linear Regression value, and the Standard
  Error }
var BAR: integer;
Bar := BarCount - 1;
DrawLabel( 'Linear Reg = ' + FormatFloat( '#,##0.00', LinearReg( Bar,
#Close, 30 ) ), 0 );
DrawLabel( 'Std Error = ' + FormatFloat( '#,##0.00', StdError( Bar,
#Close, 30 ) ), 0 );

```

16.69 StochD

StochD(Bar: integer; Period: integer; Smooth: integer): float;
 StochDSeries(Period: integer; Smooth: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

StochD returns the Stochastic %D indicator. StochD is a smoothed version of the Stochastic %K (see StochK). Specify the length of smoothing desired in the *Smooth* parameter. The indicator can range from 0 to 100. Values near 0 indicate that most of the recent price action closed near the days lows, and readings near 100 indicate that prices are closing near the upper range.

The Stochastic is a momentum indicator. The closing price tends to close near the high in an uptrend and near the low in a downtrend. If the closing price then slips away from the high or the low, then momentum is slowing. Stochastics are most effective in broad trading ranges or slow moving trends.

The %K and %D combination is called the fast stochastic. You can use the StochD indicator as the basis for creating a "Slow Stochastic" %K. To create a Slow Stochastic signal line, just take a moving average of the StochD.

Remarks

- StochD is not valid until Bar Number *Period + Smooth - 1*.

Interpretation

- StochD is used as a signal line for StochK. A buy is triggered when StochK crosses above StochD from a level typically below 30. A sell is triggered when StochK crosses below StochD from typically above 70.
- Ranging markets, go long on bullish divergences, especially where the first trough is below 30.
- Ranging markets, go short on bearish divergences, especially where the first peak is above 70.
- Trending market, when either Stochastic line crosses below 30 (signal day), place a stop order to go long if prices rise above the high of the signal day or any subsequent day with a lower low. Place stop order below the low of the same day.
- Trending markets, when either Stochastic line crosses above 70 (signal day), place a stop order to go short if prices falls below the low of the signal day or any subsequent day with a higher high. Place a stop loss order above the high of the same day.
- Trending markets, Use trend following indicators to exit. Can take profits on divergences, if confirmed by the trend following indicator.

Calculation

n = Number of periods, normally 5
 S = Number of smoothing intervals, normally 3
 $\%D$ = Slow Stochastic K , smoothed over S periods (not SMA smoothing)

$HH(\text{Bar}-j)$ = Highest High at $\text{Bar}-j$ over n periods
 $LL(\text{Bar}-j)$ = Lowest Low at $\text{Bar}-j$ over n periods
 $C(\text{Bar}-j)$ = PriceClose at $\text{Bar}-j$
 Σ = Summation from $j = 0$ to $S - 1$ periods
 $\text{Sum1} = \Sigma (C(\text{Bar}-j) - LL(\text{Bar}-j))$
 $\text{Sum2} = \Sigma (HH(\text{Bar}-j) - LL(\text{Bar}-j))$
 $\%D = 100 * \text{Sum1} / \text{Sum2}$

Example

```

{ Simple system based on Slow Stochastic }
var STOCHPANE, SLOWK, SLOWD, BAR: integer;
StochPane := CreatePane( 120, true, true );
SlowK := StochDSeries( 10, 3 );
SlowD := SMASeries( SlowK, 3 );
PlotSeries( SlowK, StochPane, #Purple, #Thick );
PlotSeries( SlowD, StochPane, #Black, #Thin );
for Bar := 20 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    if CrossOver( Bar, SlowK, SlowD ) then
      if GetSeriesValue( Bar - 1, SlowK ) < 20 then
        BuyAtMarket( Bar + 1, '' );
    end
  else
  begin
    if CrossOverValue( Bar, SlowK, 80 ) then
      SellAtMarket( Bar + 1, LastPosition, '' );
    end;
  end;
end;

```

16.70 StochK

StochK(Bar: integer; Period: integer): float;
 StochKSeries(Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

StochK returns the Stochastic Oscillator $\%K$. The Stochastic Oscillator measures how much price tends to close in the upper or lower areas of its trading range. The indicator can range from 0 to 100. Values near 0 indicate that most of the recent price action closed near the days lows, and readings near 100 indicate that prices are closing near the upper range.

The Stochastic is a momentum indicator. The closing price tends to close near the high in an uptrend and near the low in a downtrend. If the closing price then slips away from the high or the low, then momentum is slowing. Stochastics are most effective in broad trading ranges or slow moving trends.

Interpretation

The classic way to interpret the Stochastic is to wait for $\%K$ to reach an extreme level. A level above 70 typically indicates an overbought condition, while below 30 indicates

an oversold level. While these penetrations of extreme levels indicate a warning, the actual buy/sell signals occur when %K crosses %D (see StochD).

- Ranging markets, go long on bullish divergences, especially where the first trough is below 30.
- Ranging markets, go short on bearish divergences, especially where the first peak is above 70.
- Trending market, when either Stochastic line crosses below 30 (signal day), place a stop order to go long if prices rise above the high of the signal day or any subsequent day with a lower low. Place stop order below the low of the same day.
- Trending markets, when either Stochastic line crosses above 70 (signal day), place a stop order to go short if prices falls below the low of the signal day or any subsequent day with a higher high. Place a stop loss order above the high of the same day.
- Trending markets, use trend following indicators to exit. Can take profits on divergences, if confirmed by the trend following indicator.

Calculation

```
n      = Number of periods, normally 5
HHn    = Highest High over n periods
LLn    = Lowest Low over n periods
C      = PriceClose today
%K     = Stochastic K = 100 * ( C - LLn ) / ( HHn - LLn )
```

Example

```
{ A system based on Fast Stochastic Extreme Levels }
var STOCHPANE, BAR, P: integer;
StochPane := CreatePane( 100, true, true );
PlotSeries( StochKSeries( 14 ), StochPane, 505, #Thick );
DrawLabel( 'StochK( 14 )', StochPane );
for Bar := 14 to BarCount - 1 do
begin
  if CrossUnderValue( Bar, StochKSeries( 14 ), 20 ) then
    BuyAtMarket( Bar + 1, '' );
  if CrossOverValue( Bar, StochKSeries( 14 ), 20 ) then
    for P := 0 to PositionCount - 1 do
      if PositionActive( P ) then
        SellAtMarket( Bar + 1, P, '' );
end;
```

16.71 StochRSI

```
StochRSI( Bar: integer; Series: integer; Period: integer ): float;
StochRSISeries( Series: integer; Period: integer ): integer;
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

StochRSI is an indicator created by Tushar Chande that combines Stochastics with the Relative Strength Index. Like RSI, StochRSI cycles between overbought levels below 30 and oversold levels above 70. The StochRSI reaches these levels much more frequently than RSI, resulting in an oscillator that offers more trading opportunities. StochRSI moves within the range of 0 to 100. Unlike RSI, StochRSI frequently reaches the extreme 0 and 100 levels.

Interpretation

- Look for oversold levels below 30 and overbought levels above 70. These normally occur before the underlying price chart forms a top or a bottom. Note, you should change the levels depending on market conditions. Ensure the level lines cut across the highest peaks and the lowest troughs. During strong trends the StochRSI may remain in overbought or oversold for extended periods.
- If underlying prices make a new high or low that isn't confirmed by the StochRSI, this divergence can signal a price reversal. StochRSI divergences from price indicates very strong buy or sell signal.
- Swing Failures. If the StochRSI makes a lower high followed by a downside move below a previous low, then a Top Swing Failure has occurred, sell signal. If the StochRSI makes a higher low followed by an upside move above a previous high, then a Bottom Swing Failure has occurred, buy signal.
- The mid point level of 50 will often act as support or resistance if the StochRSI bounces off the 50 level. Crosses of the 50 level can be used as a buying or selling signal. When StochRSI crosses above then buy, when StochRSI crosses below then sell.

Calculation

StochRSI is essentially a StochK of the RSI. See both StochK and RSI for more information.

$$\text{StochRSI} = \left(\text{RSI}(n) - \text{RSI lowest low}(n) \right) / \left(\text{RSI highest high}(n) - \text{RSI lowest low}(n) \right)$$

where, n = number of periods

Example

```
var STOCHRSIPANE, BAR: integer;
StochRSIPane := CreatePane( 75, TRUE, TRUE );
PlotSeries( StochRSISeries( #Close, 14 ), StochRSIPane, 411, 2 );

InstallBreakEvenStop( 10 );
for Bar := 31 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  if LastPositionActive then
  begin
    if StochRSI( Bar, #Close, 14 ) = 100 then
      SellAtMarket( Bar + 1, LastPosition, '' );
    end
  else
  begin
    if EMA( Bar, #Close, 30 ) > EMA( Bar - 1, #Close, 30 ) then
      if StochRSI( Bar, #Close, 14 ) = 0 then
        BuyAtMarket( Bar + 1, '' );
      end;
    end;
  end;
```

16.72 Sum

Sum(Bar: integer; Series: integer; Period: integer): float;
SumSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the sum of values from the specified Price Series over the desired period. This is not really an indicator per se, but a mathematical function used to summate values in a price series. You can use this function to build your own custom indicators, like the example shown below.

In this example, we add all the highs for twenty bars, then add all the lows for twenty bars, producing two floating point values, xUp and xDown. The two floats are subtracted and the result used to build a new series called UpMinusDown. This new indicator series behaves similar to the ATR indicator.

Interpretation

Sum is most useful when making your own custom indicators to integrate over a specified number of *Periods*.

Calculation

Simply the addition of price over the period specified.

$$\text{Sum} = (P_1 + P_2 + \dots + P_n)$$

where,

Sum = summation of price values

P = price series to be summated, **#Open**, **#Close**, TrueRangeSeries, SMASeries, etc

...

n = number of periods or Bars

Example

```
{ Plot 20 bar sum of Highs minus Lows }
var XUP, XDOWN: float;
var UPMINUSDOWN, BAR, UDPANE: integer;
UpMinusDown := CreateSeries;
for Bar := 20 to BarCount - 1 do
begin
  xUp := Sum( Bar, #High, 20 );
  xDown := Sum( Bar, #Low, 20 );
  SetSeriesValue( Bar, UpMinusDown, xUp - xDown );
end;
UDPane := CreatePane( 70, false, true );
PlotSeries( UpMinusDown, UDPane, #Olive, #Thick );
```

16.73 TII

TII(Bar: integer; Series: integer; Period: integer; MAPeriod: integer): float;
 TIIseries(Series: integer; Period, MAPeriod): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

TII is the Trend Intensity Index. It measures the strength of a trend by tabulating the deviation of price and its moving average. Specify the number of bars to use when calculating the indicator in the *Period* parameter, and the length of the moving average to use in the *MAPeriod* parameter.

TII compares the price to its *MAPeriod* moving average, recording the deviation at each bar. If price is above the moving average, a positive deviation is recorded, and if price is below the moving average a negative deviation. The deviation is simply the distance between price and the moving average.

Once the deviations are calculated, TII is calculated as:

$$\left(\text{Sum of Positive Dev} \right) / \left(\left(\text{Sum of Positive Dev} \right) + \left(\text{Sum of Negative Dev} \right) \right) * 100$$

Interpretation

TII moves between 0 and 100. A strong uptrend is indicated when TII is above 80. A strong downtrend is indicated when TII is below 20.

Example

```
var TIIPane, TIISer, Per, MAPer: integer;
Per := 30;
MAPer := 60;
TIISer := TIISeries( #Close, Per, MAPer );
TIIPane := CreatePane( 75, true, true );
PlotSeriesLabel( TIISer, TIIPane, 009, #Thin,
  'TIISer=TII(#Close,' + IntToStr(Per) + ',' + IntToStr(MAPer) + ')' );
```

16.74 TRIX

TRIX(Bar: integer; Series: integer; Period: integer): float;
 TRIXSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

TRIX displays the percentage Rate of Change (see ROC) of a triple exponentially-smoothed moving average (EMA) over the specified *Period*. TRIX oscillates above and below the zero value. The indicator applies triple smoothing in an attempt to eliminate insignificant price movements within the trend that you're trying to isolate.

Interpretation

TRIX generates a signal when it changes direction (turns up or down). Alternately, you can create a signal line using a moving average and wait until TRIX crosses this signal line.

Note: UseUpdatedEMA affects the calculation of EMA-based indicators such as **TRIX** at runtime. Choose the default method for calculating the EMA exponent in the *Indicator Calculations* section of the Options dialog.

Example

```
{ Buy when TRIX turns up from below zero. Sell when TRIX crosses above zero. }
var TRIXPANE, BAR, Per: integer;
Per := 24;
TRIXPANE := CreatePane( 50, true, true );
PlotSeries( TRIXSeries( #Close, Per ), TRIXPANE, 520, #Thick );
DrawLabel( 'TRIX( Close, 24 )', TRIXPANE );
for Bar := 60 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    if TurnUp( Bar, TRIXSeries( #Close, Per ) ) then
      if TRIX( Bar, #Close, 24 ) < 0 then
        BuyAtMarket( Bar + 1, '' );
    end
  else
```

```

begin
  if CrossOverValue( Bar, TRIXSeries( #Close, Per ), 0 ) then
    SellAtMarket( Bar + 1, LastPosition, '' );
  end;
end;

```

16.75 Trough

Trough(Bar: integer; Series: integer; Reversal: float): float;
 TroughSeries(Series: integer; Reversal: float): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the value of the last Trough that was identified for the specified Price *Series* as of the specified *Bar*. The *Reversal* parameter determines how much of a percentage (default) or point advance is required to trigger a new Trough. It typically requires a few bars of upward price movement to reach the *Reversal* level and qualify a new Trough. The Trough function never "looks ahead" in time, but always returns the Trough value as it would have been determined as of the specified *Bar*. For this reason, the return value of the Trough function will lag, and report troughs a few bars later than they actually occurred in hindsight. This is intentional, and allows peak/trough detection to be used when back-testing trading systems.

Interpretation

(See [Peak](#)^[220])

Remarks

- To base reversals on point/absolute movement, pass the **#AsPoint** constant to the **SetPeakTroughMode** function in your script.
- Calculating peaks/troughs based on *percentage* moves is not allowed on data series that contains negative or zero values. For these data series you must use **SetPeakTroughMode** to base the reversal amount on a point value.

Calculation

Troughs are detected by looking for a percentage (default) or point reversal in the Price Series greater than or equal to the reversal amount specified in the *Reversal* parameter. For example, if you specify a reversal value of 10, and prices make a new low of \$20, a trough will be triggered at that bar as soon as prices move up to \$22 (provided they do not continue below \$20). The move up to \$22 may take several bars. During these bars the Trough function will not return \$20, but will instead return the value of the previous trough. This is because you would not have known that that \$20 was an actual trough yet because the *Reversal* level has not been met. The new Trough value of \$20 will be returned only after prices have reached the \$22 level, and the reversal level is reached.

Example

```

{ Draw the level of 7% Troughs on the chart }
var PS: integer;
PS := TroughSeries( #Low, 7 );
PlotSeries( PS, 0, #Green, #Dots );

```

16.76 TroughBar

TroughBar(Bar: integer; Series: integer; Reversal: float): integer;
TroughBarSeries(Series: integer; Reversal: float): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the bar number at which the last Trough that was identified for the specified Price *Series* as of the specified *Bar*. The *Reversal* parameter determines how much of a percentage (default) or point decline is required to trigger a new Trough. It typically requires a few bars of upward price movement to reach the Reversal level and qualify a new Trough. The Trough function never "looks ahead" in time, but always returns the Trough value as it would have been determined as of the specified bar. For this reason, the return value of the Trough function will lag, and report troughs a few bars later than they actually occurred in hindsight. This is intentional, and allows peak/trough detection to be used when back-testing trading systems.

Interpretation

(See [Peak Bar](#)^[227])

Remarks

- **TroughBar** returns -1 if a trough has not yet been detected at the beginning of the chart.
- To base reversals on point/absolute movement, pass the **#AsPoint** constant to the **SetPeakTroughMode** function in your script.
- Calculating peaks/troughs based on *percentage* moves is not allowed on data series that contains negative or zero values. For these data series you must use **SetPeakTroughMode** to base the reversal amount on a point value.

Calculation

(See [Trough](#)^[239])

Example

```
{ Flag bars that are 5% Troughs }
var Bar, n, nPrev: integer;
for Bar := 0 to BarCount - 1 do
begin
  n := TroughBar( Bar, #Low, 5 );
  if ( n <> nPrev ) and ( n > -1 ) then
  begin
    DrawCircle( 6, 0, n, PriceLow( n ), #Green, #Thick );
    nPrev := n;
  end;
end;
```

16.77 TroughNum

TroughNum(Bar: integer; Series: integer; Number: integer; Reversal: float): float;
TroughNumSeries(Series: integer; Number: integer; Reversal: float): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the value of the "Nth" most recent Trough that was identified for the specified

Price *Series* as of the specified *Bar*. The *Reversal* parameter determines how much of a percentage (default) or point decline is required to trigger a new Trough. It typically requires a few bars of upward price movement to reach the Reversal level and qualify a new Trough. The Trough function never "looks ahead" in time, but always returns the Trough value as it would have been determined as of the specified bar. For this reason, the return value of the Trough function will lag, and report troughs a few bars later than they actually occurred in hindsight. This is intentional, and allows peak/trough detection to be used when back-testing trading systems.

Use the *Number* parameter to specify which Trough to identify. To obtain the most recent Trough, pass 0 (although this is the same as using the Trough function). *Number* = 1 returns the previous Trough, 2 returns the second most previous, etc.

Interpretation

- Peaks/Troughs of highs and lows are often used as support and resistance levels. These points have historical significance because they have proven to be important levels of price reversal.
- Peaks and troughs are a convenient way of detecting chart patterns. For example, one aspect of a Head & Shoulders Top is a series of 3 Peaks, the second higher than the outer two.

Remarks

- To base reversals on point/absolute movement, pass the **#AsPoint** constant to the **SetPeakTroughMode** function in your script.
- Calculating peaks/troughs based on *percentage* moves is not allowed on data series that contains negative or zero values. For these data series you must use **SetPeakTroughMode** to base the reversal amount on a point value.

Calculation

Troughs are detected by looking for a percentage (default) or point reversal in the Price Series greater than or equal to the reversal amount specified in the *Reversal* parameter. For example, if you specify a reversal value of 10, and prices make a new low of \$20, a trough will be triggered at that bar as soon as prices move up to \$22 (provided they do not continue below \$20). The move up to \$22 may take several bars. During these bars the Trough function will not return \$20, but will instead return the value of the previous trough. This is because you would not have known that that \$20 was an actual trough yet because the reversal level has not been met. The new Trough value of \$20 will be returned only after prices have reached the \$22 level, and the reversal level is reached.

Example

```
{ Draw resistance necklines for potential Double Bottoms }
var T1, T2, DIFF, DIFFPCT, P1: float;
var LASTPEAK: float;
var BAR: integer;
for Bar := 120 to BarCount - 1 do
begin
  if ROC( Bar, #Close, 120 ) < -20 then
  begin
    t1 := TroughNum( Bar, #Low, 0, 5 );
    t2 := TroughNum( Bar, #Low, 1, 5 );
    Diff := Abs( t1 - t2 );
    DiffPct := ( Diff / PriceClose( Bar ) ) * 100;
    if DiffPct < 5 then
    begin
```

```

    p1 := Peak( Bar, #High, 5 );
    if p1 <> LastPeak then
    begin
        LastPeak := p1;
        DrawLine( Bar, p1, Bar + 60, p1, 0, #Blue, #Thin );
    end;
end;
end
end;
end;

```

16.78 TrueRange

TrueRange(Bar): float;
TrueRangeSeries: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

True Range is apart of the Welles Wilder indicator collection. It is the actual range, high to low, of a bar. It includes any gap, between today's High or Low and yesterday's Close. As it can use the previous day in its calculation, the first periods true range may be undefined. The True Range is the maximum price movement over a period. True Range is the basis of the Average True Range (see ATR) indicator.

Interpretation

- True Range is a way to express a daily range without ignoring gaps that can occur between the previous close and the open.
- The True Range is not intended to be used as an indicator.
- The True Range is useful if used as a Price Series Parameter for another indicator as in the EMA volatility example show below.
- Especially useful in volatility indicators, where the high and low prices may not include the full volatility of price action.
- Can be incorporated in entry or exit triggers.

Calculation

True Range is always a positive number and is defined by Welles Wilder to be the greatest of the following for each period:

- The distance from today's high to today's low.
- The distance from yesterday's close to today's high.
- The distance from yesterday's close to today's low.

Example

```

{ Example 1, Show how the ATR indicator is created }
var MYATR, ATRPANE: integer;
MyATR := WilderMASeries( TrueRangeSeries, 14 );
ATRPane := CreatePane( 100, true, true );
PlotSeries( MyATR, ATRPANE, #Maroon, #Thick );
PlotSeries( ATRSeries( 14 ), ATRPANE, #Red, #Thin );
DrawLabel( 'Ex 1: WilderMASeries(TrueRangeSeries,14)', ATRPANE );

{ Example 2, Show how True Range is used in EMA volatility }
var TRPANE: integer;
var range_s: integer; // Series
// EMA(True Range)
range_s := EMASeries(TrueRangeSeries, 21);

```

```
// Plot True Range
TRPane := CreatePane( 100, true, true );
PlotSeries( range_s, TRPane, #Maroon, #Thick );
DrawLabel( 'Ex 2: EMA(TrueRangeSeries)', TRPane );
```

16.79 UltimateOsc

UltimateOsc(Bar: integer): float;
UltimateOscSeries: integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Williams' Ultimate Oscillator uses weighted sums of three oscillators, each using a different time period (7, 14, and 28), which represent short, medium, and long term market trends. The Ultimate Oscillator moves within the range of 0 to 100.

Interpretation

Williams recommended method of interpreting the Ultimate Oscillator is to look for divergences between the indicator value and price. For example, a bullish divergence occurs when prices make a lower low, but the Ultimate Oscillator fails to make a lower low.

Example

```
{ Look for bullish divergences between Ultimate Oscillator and Price }
var T1, T2: float;
var UUP, PUP: boolean;
var ULTOSCPANE, UTROUGH, PTROUGH, BAR: integer;

UltOscPane := CreatePane( 80, true, true );
PlotSeries( UltimateOscSeries, UltOscPane, 022, #Thick );
DrawLabel( 'UltimateOsc', UltOscPane );

SetPeakTroughMode( #AsPoint );
UTrough := TroughSeries( UltimateOscSeries, 10 );
PlotSeries( UTrough, UltOscPane, #Green, #Dots );
PTrough := TroughSeries( #Close, 10 );
PlotSeries( PTrough, 0, #Green, #Dots );
for Bar := 100 to BarCount - 1 do
begin
  t1 := Trough( Bar, UltimateOscSeries, 10 );
  t2 := TroughNum( Bar, UltimateOscSeries, 1, 10 );
  UUp := t1 > t2;
  t1 := Trough( Bar, #Close, 10 );
  t2 := TroughNum( Bar, #Close, 1, 10 );
  PUp := t1 > t2;
  if UUp and not PUp then
    SetBarColor( Bar, #Green );
end;
```

16.80 VHF

VHF(Bar: integer; Series: integer; Period: integer): float;
VHFSeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

The Vertical Horizontal Filter is used to determine if prices are trending or are in a congestion stage. It is calculated by dividing the difference in the sums of highest and lowest values by the sum of the absolute values of daily price differences. Typically a period of 28 is used for VHF.

Interpretation

VHF describes how strongly prices are trending. The higher the VHF value, the stronger the trend.

Example

```
{ Color chart background when prices are trending according to VHF }
var VHFPane, BAR: integer;
VHFPane := CreatePane( 70, true, true );
PlotSeries( VHFSeries( #Close, 28 ), VHFPane, 952, #Thin );
DrawLabel( 'VHF( 28 )', VHFPane );
for Bar := 28 to BarCount - 1 do
    if VHF( Bar, #Close, 28 ) > 0.4 then
        SetBackgroundColor( Bar, 789 );
```

16.81 Vidya

Vidya(Bar: integer; Series: integer; VolatilityIndex: integer; Alpha: float): float;
 VidyaSeries(Series: integer; VolatilityIndex: integer; Alpha: float): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Vidya returns Tushar Chande's Variable Index Dynamic Average. Vidya is similar to an Exponential Moving Average, but uses a different period for each bar of calculation. The period to use is determined by the bar's volatility. Bars with a high volatility will use a shorter period, and those with a low volatility a longer period. This results in an indicator that is very responsive to abrupt market moves, but becomes less responsive during consolidation periods.

Vidya requires a "Volatility Index", which should be some indicator that tracks market volatility. Specify the Volatility Index as a Price Series in the *VolatilityIndex* parameter. The resulting Vidya will have lower effective periods on bars where the Index indicates high volatility.

Vidya also requires a floating point *Alpha* parameter. The values of the Volatility Index are multiplied by Alpha to modulate the sensitivity of the Vidya. You should arrive at a value of Alpha such that $(1 - \text{Alpha} * \text{Volatility Index})$ never becomes negative.

Interpretation

You can interpret Vidya as you would another moving average. Additionally, Vidya tends to respond more quickly and go flat during consolidation periods.

Example

```
{ Vidya using Standard Deviation as Volatility Index }
var STD, SMASD, VOLIDX, VIDYASTDDEV: integer;

{ Obtain Current Std Dev as a ratio of Historic Std Dev }
Std := StdDevSeries( #Close, 10 );
SmaStd := SMASeries( Std, 50 );
VolIDX := DivideSeries( Std, SmaStd );
```

```

{ Create the Vidya }
VidyaStdDev := VidyaSeries( #Close, VolIDX, 0.1 );
PlotSeries( VidyaStdDev, 0, #Red, 2 );

```

16.82 VMA

VMA(Bar: integer; Series: integer; Period: integer): float;
 VMASeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

VMA returns the Volume-Weighted Moving Average for the specified Price *Series* and *Period*. VMA is similar to a Simple Moving Average (SMA), but each bar of data is weighted by the bar's volume.

VMA places more significance on bars with the largest volume and less for bars with lowest volume for the *Period* specified. The VMA value attempts to represent the average purchase price of the past number of periods, as it assumes that all prices were traded at selected time (usually the closing value). Using VMA, you can judge if you are buying at a low value or selling at a high value compared to the averaged price paid by all market participants.

Because important breakouts are often accompanied by a large increase in volume, VMA will track aggressive moves more closely than other types of moving averages. During consolidation periods, where volume is light, VMA will act like a normal Simple Moving Average.

Interpretation

- Use the same rules that we apply to SMA when interpreting VMA. Keep in mind, though, that VMA is generally more sensitive to price movement on high volume days.
- VMA's are used to determine trend direction. If the VMA is moving up, the trend is up, if moving down then the trend is down. A 200-bar VMA is common proxy for the long term trend. 60-bar VMA's are typically used to gauge the intermediate trend. Shorter-period VMA's can be used to determine short-term trends.
- VMA's are commonly used to smooth price data and technical indicators. Applying a VMA smooths out choppy data. The longer the period of the VMA, the smoother the result, but more lag is introduced between the VMA and the source series.
- VMA crossing price is often used to trigger trading signals. For example, when prices cross above the VMA go long, and when they cross below the VMA go short.
- Look for differences between the SMA and the VMA with the same number of periods. When the WMA is above the SMA then buyers are active and are accumulating stock, go long. When the WMA is below the SMA then seller are active, and stock is being sold, go short.

Calculation

$$VMA = (V1 * P1 + V2 * P2 + \dots + Vn * Pn) / (V1 + V2 + \dots + Vn)$$

where,

P1 = current price
 P2 = price one bar ago, etc . .
 V1 = current volume
 V2 = volume one bar ago

n = number of periods/bars

Example

```
{ Compare a Volume Weighted Moving Average with a standard MA }
PlotSeries( SMASeries( #Close, 60 ), 0, 005, #Thin );
PlotSeries( VMASeries( #Close, 60 ), 0, 005, #Thick );
```

16.83 Volatility

Volatility(Bar: integer; Period: integer; ROCPeriod: integer): float;
VolatilitySeries(Period: integer; ROCPeriod: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Return's Chaikin's Volatility indicator of the specified Moving Average and ROC periods. Chaikin's Volatility first calculates an Exponential Moving Average (EMA) of the difference between the High and Low price. The Volatility indicator is then created by taking the Rate of Change (ROC) of this value over a period specified by *ROCPeriod*. The period of the EMA is specified in the *Period* parameter.

Interpretation

- High values indicate that intraday prices have a wide high to low range. Low values indicate that intraday prices have relatively constant high to low range.
- Market tops accompanied by increase volatility over short periods of time, indicate nervous and indecisive traders. Or market tops with decreasing volatility over long time frames, indicate maturing bull markets.
- Market bottoms accompanied by decreased volatility over long periods of time, indicate bored and disinterested traders. Or market bottoms with increasing volatility over relatively sort time periods, indicate panic sell off.

Calculation

First calculate an Exponential Moving Average (EMA) of the difference between High and Low price.

$$HLAve = 10\text{-day EMA}(High - Low)$$

Then take the Rate of Change (ROC) of this value over a period specified by *ROCPeriod*.

$$CV = (HLAve) / (HLAve \text{ n days ago})$$

where,

CV = Chaikin's Volatility value
n = number of ROC periods

Note: UseUpdatedEMA affects the calculation of EMA-based indicators such as **Volatility**. Choose the default method for calculating the EMA exponent in the *Indicator Calculations* section of the Options dialog.

Example

```
{ Colors bars with higher intensity red as volatility increases }
var V: float;
var VOLPANE, BAR, N: integer;
VolPane := CreatePane( 100, true, true );
PlotSeries( VolatilitySeries( 14, 10 ), VolPane, 062, #ThickHist );
```

```

DrawLabel( 'Volatility( 14, 10 )', VolPane );
for Bar := 0 to BarCount - 1 do
begin
  v := Volatility( Bar, 14, 10 );
  n := Round( v ) div 10;
  if n < 0 then
    n := 0;
  if n > 9 then
    n := 0;
  SetBarColor( Bar, n * 100 );
end;

```

16.84 WilderMA

WilderMA(Bar: integer; Series: integer; Period: integer): float;
 WilderMASeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

WilderMA is sometimes call *Wilder's Smoothing*, and it returns the Moving Average as calculated by Welles Wilder in his book *New Concepts in Technical Trading*. This indicator is similar to the Exponential Moving Average. Compared to other moving averages, WildersMA responds slowly to price changes. A n-period WildersMA gives similar values to a 2n period EMA. For example, a 14-period EMA has almost the same values as a 7-period WilderMA.

Interpretation

- WilderMA can be interpreted in the same way as other moving averages. The WilderMA is like a EMA with half number of periods. See the EMA indicator for more information.
- You should use a WilderMA when calculating other Wilder's indicators to ensure consistent results with other systems and users.
- If you are after a smoothing indicator for general use, it is best to use the **SMA** or **EMA**.

Calculation

WilderMA is calculated for periods "n" as follows:

$$\text{Wilder MA} = (\text{Previous Wilder MA} * (n - 1) + \text{Price Series Value}) / n$$

where,

n = number of periods

Price Series Value = data you wish to average

Example

```

{ Compare a Simple and Exponential Moving Average with Wilder's MA }
var n: integer;
var s, s2: string;
n := 14;
s := '(Close, ' + IntToStr( n ) + ' )';
s2 := '(Close, ' + IntToStr( 2 * n ) + ' )';
PlotSeriesLabel( WilderMASeries( #Close, n ), 0, 090, #Thick,
'WilderMA' + s );
PlotSeriesLabel( SMASeries( #Close, n ), 0, 900, #Thin, 'SMA' + s );
PlotSeriesLabel( EMASeries( #Close, 2 * n ), 0, 005, #Thin, 'EMA' + s2
);

```

16.85 WilliamsR

WilliamsR(Bar: integer; Period: integer): float;
WilliamsRSeries(Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Williams %R is a momentum indicator developed by Larry Williams. Like Stochastic Oscillators (StochK, StochD), WilliamsR is used to gauge overbought and oversold levels, and ranges between 0 and 100. However, unlike most other momentum oscillators, the low end of the scale represents an overbought area, and the high end an oversold condition. For this reason Williams %R is often multiplied by -1 and plotted on a negative scale.

Williams %R measures the latest closing price relative to high low range within the past data, thus it reflects buyers and sellers commitment to close the price within that range. At the peak of the buyer's power the oscillator reaches zero, and at the peak of the seller's power, it reaches 100. The overbought region is below 10 percent and the oversold region is over 90 percent.

Interpretation

A reading of above 80 or 90 indicates oversold levels, and below 20 or 10 indicates overbought. Williams %R has a tendency to peak ahead of price, so it can be a good tool in identifying trend reversals. During strong trends, the Williams %R can remain in the oversold or overbought regions for extended periods of time.

- In ranging markets, go long when the indicator falls below the oversold line then rises back above the oversold line.
- In ranging markets, go short when indicator rises above the overbought line the falls back below the overbought line.
- In ranging markets, go long on bullish divergences, if the indicator's first trough is in the oversold zone.
- In ranging markets, go short on bearish divergences, if the indicator's first peak is in the overbought zone.
- In a up trend or rally, go short if the indicator fails to reach the oversold region and begins to fall. This is a swing failure, it show the buyers are weakening.
- In a down trend, go long when the indicator fails to reach the overbought region and begins to rise. This is a swing failure, it show the sellers are weakening.

Calculation

$$Wm\%R = 100 * (H_n - C) / (H_n - L_n)$$

where,

n = period, such as 7 days
H_n = Highest high in last n periods
L_n = Lowest low in last n period
C = closing price of latest bar

Example

```
{ Color the chart background for smoothed Williams %R oversold and
overbought levels }
var X: float;
var PCTRPANE, SMOOTHR, BAR: integer;
```

```

PctRPane := CreatePane( 75, true, true );
PlotSeries( WilliamsRSeries( 14 ), PctRPane, 511, #Thick );
SmoothR := WilderMASeries( WilliamsRSeries( 14 ), 4 );
DrawLabel( 'WilliamsR( 14 )', PctRPane );
PlotSeries( SmoothR, PctRPane, #Black, #Thin );
for Bar := 20 to BarCount - 1 do
begin
  x := GetSeriesValue( Bar, SmoothR );
  if x < 20 then
    SetBackgroundColor( Bar, #RedBkg )
  else if x > 80 then
    SetBackgroundColor( Bar, #BlueBkg );
end;

```

16.86 WMA

WMA(Bar: integer; Series: integer; Period: integer): float;
 WMASeries(Series: integer; Period: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

WMA returns a linearly-Weighted Moving Average of the Price *Series* over the specified *Period*.

Whereas a Simple Moving Average (SMA) calculates a straight average of the data, WMA applies more weight to the data that is more current. The most weight is placed on the most recent data point. Because of the way it's calculated, WMA will follow prices more closely than a corresponding SMA.

Interpretation

- Use the same rules that we apply to SMA when interpreting WMA. Keep in mind, though, that WMA is generally more sensitive to price movement. This can be a double-edged sword. On the one hand, it can get you into trends a bit earlier than an SMA would. On the other hand, the WMA will probably experience more whipsaws than a corresponding SMA.
- Use the WMA to determine trend direction, and trade in that direction. When the WMA rises then buy when prices dip near or a bit below the WMA. When the WMA falls then sell when prices rally towards or a bit above the WMA.
- Moving averages can also indicate support and resistance areas. A rising WMA tends to support the price action and a falling WMA tends to provide resistance to price action. This reinforces the idea of buying when price is near the rising WMA or selling when price is near the falling WMA.
- All Moving Averages, including the WMA are not designed to get you into a trade at the exact bottom and out again at the exact top. They tend to ensure that you're trading in the general direction of the trend, but with a delay at the entry and exit. The WMA has a shorter delay than the SMA.

Calculation

WMA is a *linearly-weighted* moving average that is calculated by multiplying the first data point (oldest in time) by 1, the second by 2, the third by 3, etc. The final result is then divided by the sum of the weights. More recent data is thus more heavily weighted, and contributes more to the final WMA value. WMA excludes price data outside the length of the moving average, *Period*.

$$\text{WMA} = (P_1 * n + P_2 * (n-1) + P_3 * (n-2) + \dots) / (n + (n-1) + (n-2))$$

+ ...)

where,

P1 = Current Price

P2 = price one bar ago, etc....

n = number of periods

Example

```
{ This sample system acts on crossovers of 60 and 80 period WMAs }
var BAR, WMASlow, WMAFast: integer;
WMASlow := WMAseries( #Close, 80 );
WMAFast := WMAseries( #Close, 60 );
PlotSeriesLabel( WMAFast, 0, 900, #Thin, 'WMAFast' );
PlotSeriesLabel( WMASlow, 0, 009, #Thin, 'WMASlow' );

InstallProfitTarget( 20 );
for Bar := 80 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  if LastPositionActive then
  begin
    if CrossUnder( Bar, WMAFast, WMASlow ) then
      SellAtMarket( Bar + 1, LastPosition, '' );
    end
  else
  begin
    if CrossOver( Bar, WMAFast, WMASlow ) then
      BuyAtMarket( Bar + 1, '' );
    end;
  end;
end;
```

17 Time Frame Functions

17.1 Overview

The Time Frame functions are probably the most difficult to understand of the WealthScript functions, yet once you have mastered them, you will see how easy it is to create complex trading systems based on data and indicators in other time frames.

Two concepts relating to time frames are necessary to understand. The first is that you can **Scale** the data in the primary series using the using the Scale toolbar for ChartScripts ([DWM, 5, 3](#)) and the Scale tab controls in the \$imulator, Rankings, and Scans tools. Scaling in this manner re-creates the data into a new base time frame, which allows you to generate trades in the new scale. Note that the [ChangeScale](#)^[251] function serves this same purpose, but it is useful only in the ChartScript window.

Unlike the aforementioned scaling features of Wealth-Lab, the Time Frame functions do not change the base time frame and therefore do not allow you to make trades on resultant Price Series. This group of functions allow you to create indicators in more compressed time frames that must be *restored* or *projected* back to the original base time frame.

Scaling and Time Frame Notes:

1. Transforming intraday data to multiples of its underlying interval using the Scale toolbar is currently available only for ChartScript windows. A similar *intraday scaling* feature does not exist for the \$imulator, Scans, and Rankings.
2. It is not possible to place trades on a Primary Series that has been time-compressed *from within a script* using [SetScaleCompressed](#) or [SetScaleDaily](#), for example. These WealthScript Time Frame functions allow you only to generate indicators and other Price Series in a more compressed time frame that must be referenced back to the base time frame.

For more information, see the discussion of Understanding Time Frames in the WealthScript Guide.

Note: The Time Frame category of WealthScript functions are not available for SimuScripts.

17.2 ChangeScale

ChangeScale(Scale: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Changes the scale of the chart window to the scale specified in the *Scale* parameter. You must use the following constants to specify scale:

#Daily, #Weekly, #Monthly

Use this function if you have a system that should always operate on weekly data, for example, to save you from having to manually change scale from the toolbar.

Remarks

- **Available from the ChartScript window only.**
- **ChangeScale** works differently than other functions. The parser looks for the statement in the code and changes scale before executing the script. If you do not want to change the scale then do not include this statement - even in a comment block!
- Do not use **ChangeScale** more than once in the same ChartScript. Only the first **ChangeScale** call will be honored.

Example

```
ChangeScale( #Weekly );
{ System rules for weekly scale only follow }
```

17.3 DailyFromMonthly

DailyFromMonthly(Series: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns the daily price *Series* [handle] from a previously accessed monthly series (using **SetScaleMonthly**) specified by the parameter *Series*. The function creates a new Price Series synched with the current daily chart, and populates it with the appropriate values from the monthly series. The result will be repeated values (typically 20 or so) for each month within the series.

Example

```
{ Plot monthly high/low bands on the daily chart }
var MH, ML, MHP, MLP: integer;
SetScaleMonthly;
mh := #High;
ml := #Low;
RestorePrimarySeries;
mhp := DailyFromMonthly( mh );
mlp := DailyFromMonthly( ml );
PlotSeries( mhp, 0, #Red, #Thin );
PlotSeries( mlp, 0, #Blue, #Thin );
```

17.4 DailyFromWeekly

DailyFromWeekly(Series: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns a daily price series [handle] from a previously accessed weekly series (using **SetScaleWeekly**) specified by the parameter *Series*. The function creates a new Price Series synched with the current daily chart, and populates it with the appropriate values from the weekly series. The result will be repeated values (typically 5) for each week within the series.

Example

```
{ Plot 52 week moving average on the daily chart }
var WSMA, WSMAP: integer;
SetScaleWeekly;
```

```
wsma := SMAseries( #Close, 52 );
RestorePrimarySeries;
wsmap := DailyFromWeekly( wsma );
PlotSeries( wsmap, 0, #Olive, #Thick );
```

17.5 GetDailyBar

GetDailyBar(Bar: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Lets you access daily data from within an intraday chart. The daily series must have been obtained after a call to **SetScaleDaily**. **GetDailyBar** returns the Bar Number of the daily series that corresponds to the specified *Bar* in the intraday series. This function works only within an intraday chart.

Example

```
{ Highlight bars on the intraday chart where a
  daily moving average crossover took place }
var Bar, BarDaily, SMA1, SMA2: integer;

{ Get the daily moving averages }
SetScaleDaily;
SMA1 := SMAseries( #Close, 5 );
SMA2 := SMAseries( #Close, 10 );
RestorePrimarySeries;

{ Now, cycle through our intraday bars }
for Bar := 60 to BarCount - 1 do
begin
  { Get corresponding daily bar value }
  BarDaily := GetDailyBar( Bar );
  if BarDaily > 1 then
    if CrossOver( BarDaily, SMA1, SMA2 ) then
      SetBarColor( Bar, #Green );
end;
```

17.6 GetIntraDayBar

GetIntraDayBar(Bar: integer; Interval: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Note: Not applicable to second or tick-based charts.

Description

Lets you access a compressed Price Series in a higher time frame (created with **SetScaleCompressed**). This function takes a *Bar* number and converts it to the corresponding bar in the compressed series *Interval*. So you can, for example, get the correct bar of data in a 15 minute compressed Price Series from a 1 minute chart.

Example

```
{ Uses 15 minute RSI for buy/sell signals }
var RSIPANE, RSI_20_15, RSI_20_15_S, BAR, BAR15: integer;

RSIPane := CreatePane( 100, true, true );
SetScaleCompressed( 15 );
```



```

RSI_20_15 := RSISeries( #Close, 20 );
RSI_20_15 := OffsetSeries( RSI_20_15, -1 );
RestorePrimarySeries;
RSI_20_15_S := IntradayFromCompressed( RSI_20_15, 15 );
PlotSeries( RSI_20_15_S, RSIPane, #Blue, #Thin );

for Bar := 100 to BarCount - 1 do
begin
  Bar15 := GetIntraDayBar( Bar, 15 );
  if CrossOverValue( Bar15, RSI_20_15, 30 ) and not LastPositionActive
  then
    BuyAtMarket( Bar + 1, '15 minute RSI' )
  else if CrossOverValue( Bar15, RSI_20_15, 50 ) then
    SellAtMarket( Bar + 1, LastPosition, '15 minute RSI' );
end;

```

17.7 GetMonthlyBar

GetMonthlyBar(Bar: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Lets you access monthly data from within a daily chart. **GetMonthlyBar** returns the Bar Number in the monthly series that corresponds to the specified *Bar* in the daily series. The monthly series must have been obtained after a call to **SetScaleMonthly**. This function only works within a daily chart.

Example

```

{ Highlight bars on the daily chart where 5 month RSI
  is below 30 }
var BAR, MONTHLYBAR: integer;
for Bar := 30 to BarCount - 1 do
begin
  MonthlyBar := GetMonthlyBar( Bar );
  begin
    SetScaleMonthly;
    if RSI( MonthlyBar, #Close, 5 ) < 30 then
      SetBarColor( Bar, #Green );
    RestorePrimarySeries;
  end;
end;

```

17.8 GetWeeklyBar

GetWeeklyBar(Bar: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Lets you access weekly data from within a daily chart. **GetWeeklyBar** provides the bar number in the weekly series that corresponds to the specified *Bar* in the daily series. The weekly series must have been obtained after a call to **SetScaleWeekly**. This function only works within a daily chart.

Example

```

{ Highlight bars on the daily chart where weekly MACD
  has turned up }

```

```

var WEEKLYMACD, BAR, WEEKLYBAR: integer;
SetScaleWeekly;
WeeklyMACD := MACDSeries( #Close );
RestorePrimarySeries;
for Bar := 10 to BarCount - 1 do
begin
    WeeklyBar := GetWeeklyBar( Bar );
    if TurnUp( WeeklyBar, WeeklyMACD ) then
        SetBarColor( Bar, #Blue );
end;

```

17.9 IntraDayFromCompressed

IntraDayFromCompressed(Series: integer; Interval: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Note: Not applicable to second or tick-based charts.

Description

Use this function to take a Price *Series* that was created in a higher time frame having the specified *Interval* (using **SetScaleCompressed**) and create a corresponding expanded Price Series that is synchronized with the base time frame.

IntraDayFromCompressed returns the integer Price Series handle for the new synchronized series.

For example, you might have created a 15 minute compressed time frame Price Series in a 1-minute chart. The compressed series has 1 bar of data for every 15 bars in the base chart. This makes plotting the compressed series impossible.

IntraDayFromCompressed will expand the 15-minute Price Series, effectively duplicating each bar 15 times. You can now safely plot the series on your 1-minute chart.

Note: If you do not wish to plot the new series, you can use **GetIntradayBar** whose advantage is one of memory savings, which may result in faster simulations on intraday data. However, it's generally more intuitive to work with the **IntraDayFromCompressed** function.

Example

```

{ Plot the 15 minute RSI on a 1 minute chart }
var RSIPANE, RSI_20_15, RSI_20_15_S: integer;

RSIPane := CreatePane( 100, true, true );
SetScaleCompressed( 15 );
RSI_20_15 := RSISeries( #Close, 20 );
RestorePrimarySeries;
RSI_20_15_S := IntradayFromCompressed( RSI_20_15, 15 );
PlotSeries( RSI_20_15_S, RSIPane, #Blue, #Thick );

```

17.10 IntraDayFromDaily

IntraDayFromDaily(Series: integer): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns an intraday Price Series [handle] from a previously accessed daily *Series* using **SetScaleDaily**. The function creates a new price series synched with the current intraday chart, and populates it with the appropriate values from the daily series. The result will be repeated values (the number depends on the interval of the intraday chart) for each day within the series.

Note: If you do not wish to plot the new series, you can use **GetDailyBar**, whose advantage is one of memory savings, which may be helpful when running large \$imulations on intraday data. However, it's generally more intuitive to work with the **IntraDayFromDaily** function.

Example

```
{ Obtain the daily moving average from the intraday chart }
var dsma: integer;
SetScaleDaily;
dsma := SMASeries( #Close, 10 );
RestorePrimarySeries;
dsma := IntraDayFromDaily( dsma );
PlotSeries( dsma, 0, #Blue, #Thick );
```

17.11 IsDaily

IsDaily: boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true if the current chart is on daily scale.

Example

```
{ if not daily, print a message to the debug window and exit }
if not IsDaily then
begin
  Print( 'Not using daily scale' );
  exit;
end;
```

17.12 IsIntraday

IsIntraday: boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true if the current chart is on an intraday scale. An intraday scale can be made of intervals of minutes, seconds, or ticks. Use the **BarInterval** function to determine the number of minutes, ticks, or seconds per bar, i.e., the interval.

Example

```
{ if not intraday, print a message to the debug window and exit }
if not IsIntraday then
begin
  Print('Not an intraday scale');
  exit;
end;
```

17.13 IsMonthly

IsMonthly: boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true if the current chart is on monthly scale.

Example

```
{ if not monthly, print a message to the debug window and exit }
if not IsMonthly then
begin
  Print('Not using a monthly scale');
  exit;
end;
```

17.14 IsWeekly

IsWeekly: boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Returns true if the current chart is on weekly scale.

Example

```
{ if not weekly, print a message to the debug window and exit }
if not IsWeekly then
begin
  Print('Not using a weekly scale');
  exit;
end;
```

17.15 SetScaleCompressed

SetScaleCompressed(Interval: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Note: Not applicable to second or tick-based charts.

Description

Sets the chart's time scale to a more compressed intraday time frame. Future Price Series that are acquired will be based on the new time frame, which is specified in the *Interval* parameter. You should always revert back to the base scale by calling **RestorePrimarySeries** after changing time frame.

SetScaleCompressed will work on intraday charts only. You can only change to a higher time frame that is possible to create from the current time frame. For example, if you're working in a base 5-minute chart, you can compress to 10, 15 or 20 minutes, but not to 1 or 7 minutes. Base 1-minute charts can be compressed to any higher time interval.

You can use the higher time frame data series in your charts in 2 ways. To plot an entire series you must first convert the higher time frame series back to the base time frame by using **IntraDayFromCompressed**. To access an individual bar from a higher time frame series (without the ability to plot the series in the lower time frame) you should use **GetIntraDayBar** to obtain the correct bar number to access.

Important Note

When using data from a higher time frame in trading systems, you should be sure to take an action only on a bar that has complete data for the higher time frame. For example, if accessing 15-minute bars, only take trades on even 15 minute boundaries. Alternately, you can shift the higher time frame Price Series 1 bar to the right using OffsetSeries to safely use the previous value.

Example

```
{ The chart will depict a 15 minute SMA and RSI from a 1, 3 or 5 minute
chart }
var SMA_20, SMA_20_15, SMA_20_15_S, RSIPANE, RSI_20_15, RSI_20_15_S:
integer;

SMA_20 := SMASeries( #Close, 20 );
SetScaleCompressed( 15 );
SMA_20_15 := SMASeries( #Close, 20 );
RestorePrimarySeries;
SMA_20_15_S := IntradayFromCompressed( SMA_20_15, 15 );
PlotSeries( SMA_20, 0, #Red, #Thin );
PlotSeries( SMA_20_15_S, 0, #Blue, #Thin );

RSIPane := CreatePane( 100, true, true );
PlotSeries( RSISeries( #Close, 20 ), RSIPane, #Red, #Thin );
SetScaleCompressed( 15 );
RSI_20_15 := RSISeries( #Close, 20 );
RestorePrimarySeries;
RSI_20_15_S := IntradayFromCompressed( RSI_20_15, 15 );
PlotSeries( RSI_20_15_S, RSIPane, #Blue, #Thin );
```

17.16 SetScaleDaily

```
SetScaleDaily;
```

```
ChartScripts SimuScripts PerfScripts CMScripts
```

Description

Sets the chart's time scale to daily data from an intraday chart. Future Price Series that are acquired will be daily series. You should always revert back to the intraday scale by calling **RestorePrimarySeries** after changing the scale to daily.

You can use the daily data series in your intraday charts in two ways. To plot an entire series you must first convert the daily series to an intraday one with **IntraDayFromDaily**. To access an individual bar from a daily series (without the need to convert the daily series to an intraday time frame) you should use **GetDailyBar** to obtain the correct bar number to access in the daily series as shown in the example.

Example

```

{ Look for a Daily SMA CrossOver in our intraday chart }
var Bar, db, SMA1, SMA2: integer;
SetScaleDaily;
SMA1 := SMASeries( #Close, 10 );
SMA2 := SMASeries( #Close, 40 );
RestorePrimarySeries;
for Bar := 200 to BarCount - 1 do
begin
  db := GetDailyBar( Bar );
  if CrossOver( db, SMA1, SMA2 ) then
  begin
    SetBackgroundColor( Bar, #BlueBkg );
  end;
end;
end;

```

17.17 SetScaleMonthly

SetScaleMonthly;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Sets the chart's time scale to monthly data from a daily chart. Future price series that are acquired will be monthly series. You should always revert back to the daily scale by calling **RestorePrimarySeries** after changing the scale to monthly.

You can use the monthly data series in your daily charts in two ways. To plot an entire series you must first convert the monthly series to a daily one with **DailyFromMonthly**. Alternatively, to access an individual bar from a monthly series (without the need to convert the monthly series to a daily time frame) you should use **GetMonthlyBar** to obtain the correct bar number to access in the monthly series.

Example

```

{ Plot the 5 month RSI in our daily chart }
var MonthlyRSI, PlotMonthlyRSI, RSIPane: integer;
SetScaleMonthly;
MonthlyRSI := RSISeries( #Close, 5 );
RestorePrimarySeries;
PlotMonthlyRSI := DailyFromMonthly( MonthlyRSI );
RSIPane := CreatePane( 100, true, true );
PlotSeries( PlotMonthlyRSI, RSIPane, #Navy, #Thick );

```

17.18 SetScaleWeekly

SetScaleWeekly;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Sets the chart's time scale to weekly data from a daily chart. Future price series that are acquired will be weekly series. You should always revert back to the daily scale by calling **RestorePrimarySeries** after changing the scale to weekly.

You can use the weekly data series in your daily charts in two ways. To plot an entire series you must first convert the weekly series to a daily one with **DailyFromWeekly**. Alternatively, to access an individual bar from a weekly series (without the need to

convert the weekly series to a daily time frame) you should use **GetWeeklyBar** to obtain the correct bar number to access in the weekly series.

Example

```
{ Plot the weekly MACD in our daily chart }
var WeeklyMACD, PlotWeeklyMACD, MACDPane: integer;
SetScaleWeekly;
WeeklyMACD := MACDSeries( #Close );
RestorePrimarySeries;
PlotWeeklyMACD := DailyFromWeekly( WeeklyMACD );
MACDPane := CreatePane( 100, true, true );
PlotSeries( PlotWeeklyMACD, MACDPane, #Maroon, #ThickHist );
```

18 Trading System Control Functions

18.1 Overview

The Trading System functions encompass those functions that enter, exit, and split positions. You can also control automatic exits or stops without the need to program them manually.

Using [SetCommission](#)^[278] and [SetSlippage](#)^[281], you can override the default costs of commissions and slippage, which are set in the Options Dialog (F12) Trading Costs/Control tab. Finally, another group of functions allow you to further influence the sizing of positions from within your ChartScript.

Note: The Trading System category of WealthScript functions are not available for SimuScripts.

18.2 ApplyAutoStops

```
ApplyAutoStops( Bar: integer );
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Processes all installed AutoStops for the specified *Bar* number. Generally, when using AutoStops, you place the **ApplyAutoStops** statement at the beginning of the main trading loop, prior to other trading system signals.

Remarks

- AutoStops are **global** in nature. For example, only a single profit target level exists, so you cannot establish different levels for multiple positions in the same script. It's best to code your own exits manually if you want to maintain different target levels.
- AutoStops are processed for the *Bar* passed to the function. It is not necessary to pass *Bar + 1* to **ApplyAutoStops** in order to trigger Alerts, which are generated as required based on installed AutoStops.
- By design, AutoStops never exit on the same bar as entry in backtesting. For information about same-bar stops for automated trading, see the Option Dialog's *Automated Execution* topic in the User Guide.

Priority of Multiple Installed Stops

In the event that more than one AutoStop is competing to exit a trade on the same bar, Wealth-Lab takes the earliest/most pessimistic exit. For example, an Installed stop loss will be processed before all other Installed stops except the time-based exit. Specifically, the order of priority is as follows:

1. **InstallTimeBasedExit**
2. **InstallStopLoss**
3. **InstallTrailingStop**
4. **InstallReverseBreakEvenStop**
5. **InstallBreakEvenStop**
6. **InstallProfitTarget**

Mixing Manual and Installed AutoStops

Manually-coded exits execute in the order that they are coded. However, if you mix manual exits and AutoStops, manual exits that execute on *Bar + 1* will take priority unless you explicitly change the order of priority by passing *Bar + 1* to **ApplyAutoStops**. In this case, you must place the statement in the exit logic for correct operation as shown in the following example that mixes a time-based AutoStop exit with a manual profit target.

```
var Bar, p: integer;
PlotStops;
InstallTimeBasedExit( 20 );
for Bar := 10 to BarCount - 1 do
begin
  if LastPositionActive then
  begin
    ApplyAutoStops( Bar + 1 ); // Note placement within exit logic
    p := LastPosition;
    SellAtLimit( Bar + 1, PositionEntryPrice( p ) * 1.05, p, '5%
ProfitTgt' );
  end
  else
  begin
    BuyAtStop( Bar + 1, Highest( Bar, #High, 8 ), '' );
  end;
end;
```

Precedence discussion: topic?id=6315

Manually-coded exits will exit at whichever bar you specify, and in the order in which they are coded. For example, to exit with a 5% profit target on the same bar, the following code could be used instead of **InstallProfitTarget(5)**. Notice that in this case the stop is plotted on the same bar as the entry bar.

```
var Bar, p: integer;
PlotStops;
for Bar := 10 to BarCount - 1 do
begin
  { Single entry condition for demo }
  if Bar = BarCount - 20 then
    BuyAtMarket( Bar + 1, '' );

  if LastPositionActive then
  begin
    p := LastPosition;
    SellAtLimit( Bar + 1, PositionEntryPrice( p ) * 1.05, p, '5%
ProfitTgt' );
  end;
end;
```

Example

```
{ Install and execute automated stops in our trading system }
var Bar: integer;
InstallStopLoss( 8 );
InstallProfitTarget( 10 );
InstallTimeBasedExit( 40 );
for Bar := 20 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  { ... more trading system rules ... }
end;
```

18.3 BuyAtClose

BuyAtClose(Bar: integer; SignalName: string): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Establishes a long Position at market close of the specified *Bar*.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Entry Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.
- **BuyAtClose** will return boolean *false* if the signal fails to establish a new position. This can occur, for example, when using 100% equity position sizing without Leeway.

Note: AtClose orders can be difficult to realize in practice.

Example

```
{ Buy at close on a 200 bar low }
var BAR: integer;
for Bar := 200 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    if PriceLow( Bar ) = Lowest( Bar, #Low, 200 ) then
      BuyAtClose( Bar, 'Low Hit' );
    end
  else
  begin
    { .. Exit Rules ... }
  end
end;
```

18.4 BuyAtLimit

BuyAtLimit(Bar: integer; LimitPrice: float; SignalName: string): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Establishes a long Position if prices reach the indicated *LimitPrice*. The Position will be opened if prices meet or go below the specified *LimitPrice* on the specified Bar. If prices open below the *LimitPrice*, the Position will be established at open price. If prices fail to reach the *LimitPrice* objective, a Position is not established and the function returns false.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Entry Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.

- For futures symbols, *LimitPrice* is automatically rounded to the nearest Tick value.

Example

```
{ Buy the next bar if it hits the previous 10 bar low }
var X: float;
var BAR: integer;
for Bar := 200 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    x := Lowest( Bar, #Low, 10 );
    BuyAtLimit( Bar + 1, x, '10 bar low' );
  end
  else
  begin
    { .. Exit Rules ... }
  end
end;
end;
```

18.5 BuyAtMarket

BuyAtMarket(Bar: integer; SignalName: string): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Establishes a long Position at market open of the specified *Bar*.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Entry Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.
- **BuyAtMarket** will return boolean *false* if the signal fails to establish a new position. This can occur, for example, when using 100% equity position sizing without Leeway.

Example

```
{ Open a long position on the following bar based on this bar's
indicator values }
var Bar, p: integer;
for Bar := 40 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    if PriceClose( Bar ) > SMA( Bar, #Close, 40 ) then
    if BuyAtMarket( Bar + 1, 'SMA' ) then
      SetPositionPriority( LastPosition, -RSI( Bar, #Close, 14 ) );
    end
  else
  begin
    p := LastPosition;
    if Bar + 1 - PositionEntryBar( p ) = 5 then
      SellAtMarket( Bar + 1, p, 'Time-Based' );
    end;
  end;
end;
```

18.6 BuyAtStop

BuyAtStop(Bar: integer; StopPrice: float; SignalName: string): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Establishes a long Position if prices reach the indicated *StopPrice*. The Position will be opened if prices meet or exceed the specified *StopPrice* on the specified *Bar*. If prices open above the *StopPrice*, the Position will be established at open price. If prices fail to reach the *StopPrice* objective, a Position is not established and the function returns false.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Entry Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.
- For futures symbols, *StopPrice* is automatically rounded to the nearest Tick value.

Example

```
{ Try to buy a peak breakout }
var BAR: integer;
var p: float;
for Bar := 40 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    p := Peak( Bar, #Close, 15 );
    BuyAtStop( Bar + 1, p, 'Breakout' );
  end
  else
  begin
    { .. Exit Rules ... }
  end;
end;
```

18.7 CoverAtClose

CoverAtClose(Bar: integer; Position: integer; SignalName: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Covers and closes out an open short Position at closing price of the specified *Bar*.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- **CoverAtClose** will exit any position, short or long, that is passed to it in the *Position* parameter.
- To exit all open short positions pass the constant **#All** in the *Position* parameter.
- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Exit Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.

Note: AtClose orders can be difficult to realize in practice.

Example

```
{ Exit the short after 20 days }
var BAR: integer;
for Bar := 40 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    { ... Entry Rules ... }
  end
  else
  begin
    if Bar = PositionEntryBar( LastPosition ) + 20 then
      CoverAtClose( Bar, LastPosition, '20 day exit' );
    end;
  end;
end;
```

18.8 CoverAtLimit

CoverAtLimit(Bar: integer; LimitPrice: float; Position: integer; SignalName: string): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Covers and closes out an open short Position if prices meet or exceed the specified *LimitPrice*. If prices open below the *LimitPrice*, the Position is closed at market open price. If prices fail to reach the *LimitPrice* objective, the Position remains open and the function returns false. To exit all open short positions pass the constant **#All** in the *Position* parameter.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- **CoverAtLimit** can exit any position, short or long, that is passed to it in the *Position* parameter.
- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Exit Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.
- For futures symbols, *LimitPrice* is automatically rounded to the nearest Tick value.

Example

```
{ Try and exit our short position at a profit }
var BAR: integer;
var x: float;
for Bar := 40 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    { ... Entry Rules ... }
  end
  else
  begin
    x := Lowest( Bar, #Low, 10 );
    CoverAtLimit( Bar + 1, x, LastPosition, '10 day Low Limit' );
  end;
end;
```

18.9 CoverAtMarket

CoverAtMarket(Bar: integer; Position: integer; SignalName: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Used to cover and close out an open short position at the opening price of the specified *Bar*.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- **CoverAtMarket** will exit any position, short or long, that is passed to it in the *Position* parameter.
- To exit all open short positions pass the constant **#All** in the *Position* parameter.
- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Exit Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.

Example

```
{ Cover the short position if CMO becomes slightly overbought }
var BAR: integer;
for Bar := 40 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    { ... Entry Rules ... }
  end
  else
  begin
    if CMO( Bar, #Close, 20 ) > 30 then
      CoverAtMarket( Bar + 1, LastPosition, '' );
    end;
  end;
end;
```

18.10 CoverAtStop

CoverAtStop(Bar: integer; StopPrice: float; Position: integer; SignalName: string): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Used to cover and close out an open short Position if prices meet or go above the specified *StopPrice*. If prices open above the *StopPrice*, the *Position* is closed at the market open price. If prices fail to reach the *StopPrice*, the *Position* remains open and the function returns false.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- **CoverAtStop** will exit any position, short or long, that is passed to it in the *Position* parameter.
- To enter a **CoverAtStop** order for all open short positions, pass the constant **#All** in the *Position* parameter.

- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Exit Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.
- Use **PlotStops** to plot the effective stop price for *Position*.
- For futures symbols, *StopPrice* is automatically rounded to the nearest Tick value.

Example

```
{ Cover the short position if prices move against us by 10% }
var BAR: integer;
var x: float;
for Bar := 40 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    { ... Entry Rules ... }
  end
  else
  begin
    x := PositionEntryPrice( LastPosition ) * 1.1;
    CoverAtStop( Bar + 1, x, LastPosition, '10% Stop' );
  end;
end;
```

18.11 CoverAtTrailingStop

CoverAtTrailingStop(Bar: integer; Price: float; Position: integer; SignalName: string): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Implements a trailing stop at the specified *Price* level. The stop level is adjusted only if it is above the current stop level. This results in a trailing stop that is always raised, and never lowered. Otherwise, this function behaves exactly like the corresponding **CoverAtStop** function. It returns true if the stop level was reached and the position was closed.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- **CoverAtTrailingStop** will exit any position, short or long, that is passed to it in the *Position* parameter.
- To enter a **CoverAtTrailingStop** order for all open short positions, pass the constant **#All** in the *Position* parameter.
- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Exit Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.
- Use **PlotStops** to plot the effective stop price for *Position*.

Example

```
{ Initiate a trailing stop after a 5% gain }
var Bar, p: integer;

PlotStops;
for Bar := 20 to BarCount - 1 do
begin
```

```

if LastPositionActive then
begin
  p := LastPosition;
  if PositionOpenMFEPct( p, Bar ) > 5 then
    CoverAtTrailingStop( Bar + 1, SMA( Bar, #Close, 20 ), p, 'TStop'
)
  else
    CoverAtStop( Bar + 1, PositionEntryPrice( p ) * 1.10, p,
'StopLoss' );
  end
  else
    ShortAtStop( Bar + 1, Lowest( Bar, #Low, 20 ), '' );
end;

```

18.12 InstallBreakEvenStop

InstallBreakEvenStop(Trigger: float);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Installs an automatic breakeven stop once the Position reaches the specified *Trigger* profit level on a closing basis. Call **ApplyAutoStops** in your Trading System's main loop to process auto-stops.

Remarks

- *Trigger* is expressed as a percentage, points, or dollar movement as determined by the **SetAutoStopMode** function, where percentage is the default if not used.
- **InstallBreakEvenStop** is global in nature, therefore the most recent call to **InstallBreakEvenStop** will be used for subsequent trades.
- See **ApplyAutoStops** for information.

Example

```

{ Install a breakeven stop when we close above 5% profit }
var BAR: integer;
InstallBreakEvenStop( 5 );
for Bar := 20 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  { ... Entry and Exit Rules ... }
end;

```

18.13 InstallProfitTarget

InstallProfitTarget(TargetLevel: float);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Installs a profit target at the specified *TargetLevel*. Open Positions will be automatically closed if total Position profit reaches the target level. If prices gap up above the *TargetLevel* value the Position will be closed at the market open price. Call **ApplyAutoStops** in your Trading System's main loop to process AutoStops.

Remarks

- *TargetLevel* is expressed as a percentage, points, or dollar movement as determined by the **SetAutoStopMode** function, where percentage is the default if not used.
- **InstallProfitTarget** is global in nature, therefore the most recent call to **InstallProfitTarget** will be used for subsequent trades.
- For real-time automated trading, to exit with a profit on the same bar as entry use **InstallProfitTarget**. See "Automated Trading Options" in the User Guide for more information.
- See **ApplyAutoStops** for more information.

Example

```
{ If our trades ever see a 100% profit we'll be sure to cash out }
var BAR: integer;
InstallProfitTarget( 100 );
for Bar := 20 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  { ... Entry and Exit Rules ... }
end;
```

18.14 InstallReverseBreakEvenStop

InstallReverseBreakEvenStop(LossLevel: float);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Installs an automatic breakeven stop order once the Position experiences the percentage loss level specified in the *LossLevel* parameter on a closing basis. Call **ApplyAutoStops** in your Trading System's main loop to process auto-stops.

Remarks

- *LossLevel* is expressed as a percentage, points, or dollar movement as determined by the **SetAutoStopMode** function, where percentage is the default if not used.
- **InstallReverseBreakEvenStop** is global in nature, therefore the most recent call will be used for subsequent trades.
- See **ApplyAutoStops** for more information.

Example

```
{ Install a BreakEven stop to exit if we sustain losses of at least 10% }
}
var BAR: integer;
InstallReverseBreakEvenStop( 10 );
for Bar := 20 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  { ... Entry and Exit Rules ... }
end;
```

18.15 InstallStopLoss

InstallStopLoss(StopLevel: float);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Installs a stop loss at the specified *StopLevel*. Open Positions will be automatically closed if total Position loss reaches the loss value. If prices gap down below the stop loss value the Position will be closed at the market open price. Call **ApplyAutoStops** in your Trading System's main loop to process AutoStops.

Remarks

- *StopLevel* is expressed as a percentage, points, or dollar movement as determined by the **SetAutoStopMode** function, where percentage is the default if not used.
- **InstallStopLoss** is global in nature, therefore the most recent call will be used for subsequent trades.
- For real-time automated trading, to activate a stop loss exit on the same bar as entry, use **InstallStopLoss** and/or **SetRiskStopLevel**. See "Automated Trading Options" in the User Guide for more information.
- See **ApplyAutoStops** for information.

Example

```
{ Install a global automated stop loss of 15% }
var BAR: integer;
InstallStopLoss( 15 );
for Bar := 20 to BarCount - 1 do
begin
    ApplyAutoStops( Bar );
    { ... Entry and Exit Rules ... }
end;
```

18.16 InstallTimeBasedExit

InstallTimeBasedExit(Bars: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Installs an automated exit based on the specified number of *Bars*. Positions will be automatically closed after a number of bars specified in the *Bars* parameter. Be sure to call **ApplyAutoStops** in your trading system loop to activate the automated exit.

Remarks

- **InstallTimeBasedExit** is global in nature, therefore the most recent call will be used for subsequent trades.
- **AutoStops** are processed on the Bar Number passed to the function, but it is not required to pass Bar + 1 to **ApplyAutoStops** unless you wish to shift priority from manual exits to AutoStops. Changing the order precedence is necessary only when mixing **InstallTimeBasedExit** with manual exits that apply AtLimit, AtStop, or AtClose orders on Bar + 1. If you decide to give AutoStops priority over manual exits, then we recommend placing the **ApplyAutoStops(Bar + 1)** function with the rest of the exit logic as shown in the example.
- Some live feed providers may not include zero-volume bars in a [primarily intraday] chart. Since **InstallTimeBasedExit** is *bar-based*, you may wish to include manual exit logic using the **GetTime** function to achieve the desired result for sparsely-traded issues.
- See also: **ApplyAutoStops**

The example shows how to properly mix a manual exit with AutoStops that include an **InstalledTimeBasedExit**. Priority must be given to AutoStops in this [single] instance, otherwise it would allow the **SellAtStop** manual exit to trigger on the third day prior to closing the position at the open.

Example

```

{ Buy on a SMA crossover, and sell on a stop of the slow
  moving average or after 3 Bars, whichever occurs first }
var Bar, p, hSMA_S, hSMA_F, perSlow, perFast: integer;
var fStop: float;

perSlow := 20;
perFast := 10;
hSMA_S := SMA_Series( #Close, perSlow );
hSMA_F := SMA_Series( #Close, perFast );

InstallTimeBasedExit( 3 );
PlotStops;
for Bar := perSlow to BarCount - 1 do
begin
  if not LastPositionActive then
  begin { ----- Entry Rule }
    if CrossOver( Bar, hSMA_F, hSMA_S ) then
      BuyAtMarket( Bar + 1, '' );
    end
  else { ----- Exit Rules }
  begin
    { Here, installed AutoStops have priority since they are processed
    first }
    ApplyAutoStops( Bar + 1 );

    p := LastPosition;
    { Round the stop value to 2 digits after the decimal }
    fStop := Trunc( 100 * @hSMA_S[Bar] ) / 100;
    SellAtStop( Bar + 1, fStop, p, 'ManualStop' );
  end;
end;

PlotSeriesLabel( hSMA_S, 0, #Blue, #Thin, 'SMA ' + IntToStr(perSlow) );
PlotSeriesLabel( hSMA_F, 0, #Red, #Dotted, 'SMA ' + IntToStr(perFast)
);

```

18.17 InstallTrailingStop

```
InstallTrailingStop( Trigger: float; StopLevel: float );
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Installs a trailing stop to protect profits in Positions that are moving in a favorable direction. The *Trigger* value is the profit on a closing basis that the trade must show before the trailing stop is triggered, or activated. Once triggered, the stop is set to protect against a reversal of the value expressed in the *StopLevel* parameter. You must call the **ApplyAutoStops** function to actually process the stop.

Remarks

- *Trigger* is expressed as a percentage, points, or dollar movement as determined by the **SetAutoStopMode** function, where percentage is the default if not used. *StopLevel* is always expressed as a percentage reversal and remains unaffected by **SetAutoStopMode**.
- **InstallTrailingStop** is global in nature, therefore the most recent call will be used for subsequent trades.
- See **ApplyAutoStops** for information.

Example

```
{ Protect 70% of profits once we achieve 20% profit }
var BAR: integer;
InstallTrailingStop( 20, ( 100 - 70 ) );
for Bar := 20 to BarCount - 1 do
begin
  ApplyAutoStops( Bar );
  { ... Entry and Exit Rules ... }
end;
```

18.18 PortfolioSynch

PortfolioSynch(Bar: integer; Portfolio: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

PortfolioSynch creates entries and exits based on the actual Trade History for the specified *Portfolio*. Historical entry and exit signals are ignored, and the trades that appear on the chart are based on realized trades only. Consequently, this function allows you to base trading system processing on the actual signals from a specified *Portfolio*.

<i>Bar</i>	The bar being processed. Synchronization occurs on the <i>signal, or alert Bar</i> .
<i>Portfolio</i>	Specifies which Portfolio Manager portfolio to synchronize with. Pass an empty string in the <i>Portfolio</i> parameter to match against positions in any portfolio.

Remarks

- **PortfolioSynch** is functional only for Real-time ChartScript Windows and both the Scans tools (WatchList Scans and Real-Time Scans).
- When using **PortfolioSynch**, exits for the next bar (alerts) are processed only for open positions in the *Portfolio*.

ChartScript Placement

- Method 1: Call **PortfolioSynch** the first thing in your main trading system loop. This method is sufficient for most trading scripts. If your script uses **SetPositionData** or otherwise initializes local variables during the entry logic that are accessed in the exit logic, use Method 2.
- Method 2: Place **PortfolioSynch** in both the entry and exit logic, after testing for active Positions as shown in the example. It is important that **PortfolioSynch** is called once for each *Bar* in the trading loop.

Example

```

{ A Real-Time testing script that issues buys and sells every other bar
}
const MYPORT = ''; // Ensure an empty string, not a white space
var Bar: integer;
for Bar := 20 to BarCount - 1 do
begin
  // PortfolioSynch( Bar, MYPORT ); // Method 1 placement
  if LastPositionActive then
  begin
    PortfolioSynch( Bar, MYPORT ); // Method 2 placement (1 of 2)
    if Bar mod 2 = 0 then
      SellAtMarket( Bar + 1, LastPosition, '' );
    end
  else
  begin
    PortfolioSynch( Bar, MYPORT ); // Method 2 placement (2 of 2)
    if Bar mod 2 = 0 then
      BuyAtMarket( Bar + 1, '' );
    end;
  end;
end;

```

18.19 SellAtClose

SellAtClose(Bar: integer; Position: integer; SignalName: string);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Sells and closes out an open long Position at closing price of the specified *Bar*.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- **SellAtClose** will exit any position, short or long, that is passed to it in the *Position* parameter.
- To exit all open long positions pass the constant **#All** in the *Position* parameter.
- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Exit Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.

Note: AtClose orders can be difficult to realize in practice.

Example

```

{ 10 days is long enough for this system }
var BAR: integer;
for Bar := 40 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    { ... Entry Rules ... }
  end
  else
  begin
    if Bar - PositionEntryBar( LastPosition ) = 10 then
      SellAtClose( Bar, LastPosition, '10 Day Exit' );
    end;
  end;
end;

```

```
end;
```

18.20 SellAtLimit

```
SellAtLimit( Bar: integer; LimitPrice: float; Position: integer; SignalName: string ): boolean;
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Sells and closes out an open long Position on the specified *Bar* if prices meet or exceed the specified *LimitPrice*. If prices open above the *LimitPrice*, the Position is closed at market open price. If prices fail to reach the *LimitPrice* objective, the Position remains open and the function returns false. To exit all open long positions pass the constant **#All** in the *Position* parameter.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- **SellAtLimit** can exit any position, short or long, that is passed to it in the *Position* parameter.
- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Exit Signal column in the Trades View for ChartScript and \$simulator windows, or in the Signal Name column for the Scans tools.
- For futures symbols, *LimitPrice* is automatically rounded to the nearest Tick value.

Example

```
{ Try to get out at a recent high }
var BAR: integer;
var x: float;
for Bar := 40 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    { ... Entry Rules ... }
    end
  else
  begin
    x := Highest( Bar, #High, 10 ) * 1.02;
    if SellAtLimit( Bar + 1, x, LastPosition, 'Limit Sell' ) then
      Print( 'Sold!' );
    end;
  end;
end;
```

18.21 SellAtMarket

```
SellAtMarket( Bar: integer; Position: integer; SignalName: string );
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Sells and closes out an open long Position at open price of the specified *Bar*.

Remarks

- Slippage, when activated, can affect the trade's execution price.

- **SellAtMarket** can exit any position, short or long, that is passed to it in the *Position* parameter.
- To exit all open long positions pass the constant **#All** in the *Position* parameter.
- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Exit Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.

Example

```
{ Sell when prices go below the 200 day moving average }
var BAR: integer;
for Bar := 40 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    { ... Entry Rules ... }
  end
  else
  begin
    if CrossUnder( Bar, #Close, SMAseries( #Close, 200 ) ) then
      SellAtMarket( Bar + 1, LastPosition, 'Below 200 day SMA' );
    end;
  end;
end;
```

18.22 SellAtStop

SellAtStop(Bar: integer; StopPrice: float; Position: integer; SignalName: string): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Sells and closes out an open long Position if prices meet or go below the specified *StopPrice*. If prices open below the *StopPrice*, the Position is closed at market open price. If prices fail to reach the *StopPrice*, the Position remains open and the function returns false.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- **SellAtStop** can exit any position, short or long, that is passed to it in the *Position* parameter.
- To enter a **SellAtStop** order for all open long positions at *StopPrice*, simply pass the constant **#All** in the *Position* parameter.
- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Exit Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.
- Use **PlotStops** to plot the effective stop price for *Position*.
- For futures symbols, *StopPrice* is automatically rounded to the nearest Tick value.

Example

```
{ Sell at a 20% stop loss level }
var BAR: integer;
var xStop: float;
for Bar := 40 to BarCount - 1 do
begin
```

```

if not LastPositionActive then
begin
{ ... Entry Rules ... }
end
else
begin
xStop := PositionEntryPrice( LastPosition ) * 0.8;
SellAtStop( Bar + 1, xStop, LastPosition, 'Stop Loss' );
end;
end;

```

18.23 SellAtTrailingStop

SellAtTrailingStop(Bar: integer; Price: float; Position: integer; SignalName: string): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Implements a trailing stop at the specified *Price* level. The stop level is adjusted only if it is above the current stop level. This results in a trailing stop that is always raised, and never lowered. Otherwise, this function behaves exactly like the corresponding **SellAtStop** function. It returns true if the stop level was reached and the position was closed.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- **SellAtTrailingStop** will exit any position, short or long, that is passed to it in the *Position* parameter.
- To enter a **SellAtTrailingStop** order for all open long positions at the specified *Price*, simply pass the constant **#All** in the *Position* parameter.
- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Exit Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.
- Use **PlotStops** to plot the effective stop price for *Position*.

Example

```

{ Initiate a trailing stop after a 5% gain }
var Bar, p: integer;

PlotStops;
for Bar := 20 to BarCount - 1 do
begin
if LastPositionActive then
begin
p := LastPosition;
if PositionOpenMFEPct( p, Bar ) > 5 then
SellAtTrailingStop( Bar + 1, SMA( Bar, #Close, 20 ), p, 'TStop' )
else
SellAtStop( Bar + 1, PositionEntryPrice( p ) * 0.90, p,
'StopLoss' );
end
else
BuyAtStop( Bar + 1, Highest( Bar, #High, 20 ), '' );
end;
end;

```


18.24 SetAutoStopMode

SetAutoStopMode(Mode: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Allows you to control how the parameters of AutoStops are interpreted. This affects **InstallStopLoss**, **InstallProfitTarget**, **InstallTrailingStop** (first parameter only), **InstallBreakEvenStop** and **InstallReverseBreakEvenStop**.

The *Mode* parameter can be one of the following constants:

- #AsPercent** - AutoStop values are expressed as percentage moves (default)
- #AsPoint** - AutoStop values are expressed as point moves
- #AsDollar** - AutoStop values are expressed as dollar moves

Remarks

- The AutoStop *Mode* is currently a **global value** within a script execution. You cannot have some AutoStops using percent and others using point, etc.
- Remember to call **ApplyAutoStops** to execute your automated stops.

Example

```
{ Stop Loss if price declines 5% or more }
SetAutoStopMode( #AsPercent );
InstallStopLoss( 5 );

{ Stop Loss if price declines 5 points or more }
SetAutoStopMode( #AsPoint );
InstallStopLoss( 5 );

{ Stop Loss if Position declines by $2,000 or more }
SetAutoStopMode( #AsDollar );
InstallStopLoss( 2000 );
```

18.25 SetCommission

SetCommission(Commission: float);

ChartScripts SimuScripts PerfScripts CMScripts

Description

The **SetCommission** function sets the *Commission* value that is deducted from net profit every time a trade is executed. The default commission setting can be found under **Trading Costs/Control** in the Options Dialog (F12).

Note: For Wealth-Lab Developer Version 3.0, Build 4 and up, **SetCommission** no longer has any effect on setting commissions. The use of **SetCommission** in the ChartScript Window will generate an error; though in Scans, \$imulations, Rankings, and Optimizations, no error will be triggered so as not to disrupt these processes. As an alternative, use CommissionScripts for customized control of commissions. This information is included to support use of older ChartScripts that employed the function.

Remarks

- **SetCommission** has been deprecated. Use CommissionScripts to control complex commission calculations with user-defined functions.

18.26 SetPositionSize

SetPositionSize(Size: float);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Sets the Position *Size*, in dollars, that will be used for new Positions. Subsequent trades will be sized with the dollar *Size* specified.

Remarks

- Use of **SetPositionSize** is generally discouraged since SimuScripts are available to set position sizing in all tools (see **SetPositionSizeFixed**). The function is primarily for compatibility with earlier versions of Wealth-Lab.
- When using the \$imulator tool or Portfolio Simulation mode in the Position Sizing control, you must choose the radio button for **SetShare/PositionSize Value** to enable **SetPositionSize** to influence position sizing.
- In Raw Profit modes, **SetPositionSize** will override the Position Sizing control's selection. *Exception:* **SetPositionSize** has no effect in a Raw Profit WatchList or ChartScript Ranking.
- You can use both **SetPositionSize** and **SetShareSize** in the same script. As these functions are **global** in nature, the next time a trade is processed it will use the value from the function last called.

Example

```
{ Set a dynamic position size based on overbought/oversold levels }
var PS: float;
var BAR: integer;
for Bar := 40 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    ps := ( CMO( Bar, #Close, 14 ) + 200 ) * 10;
    SetPositionSize( ps );
  { ... Entry Rules ... }
  end
  else
  begin
  { ... Exit Rules ... }
  end;
end;
```

18.27 SetShareCap

SetShareCap(Cap: integer);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Sets a maximum number of shares per position to *Cap*. Ordinarily, the size of a Position is established by what you select in the Position Sizing control. However, if you use **SetShareCap** you can limit the number of shares. You can also use this feature to force a certain number of shares per Position, as follows.

Remarks

- **SetShareCap** is global in nature, therefore the most recent call to **SetShareCap** will be used for subsequent trades. This implies that in the \$imulator only the last call in the final raw-profit run of the last symbol will be used as the share cap.

Example

```
{ Force Positions to be 100 shares each }
SetPositionSize( 999999999 ); //would result in VERY large Positions
SetShareCap( 100 );           //but here we cap shares at 100
```

18.28 SetShareFloor

```
SetShareFloor( Floor: integer );
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Sets a minimum number of shares per position to *Floor*. Ordinarily, the size of a Position is established by what you select in the Position Sizing control. However, if you use **SetShareFloor** you can limit the number of shares.

Remarks

- **SetShareFloor** is global in nature, therefore the most recent call to **SetShareFloor** will be used for subsequent trades. This implies that in the \$imulator only the last call in the final raw-profit run of the last symbol will be used as the share floor.

Example

```
{ Minimum trade size of 100 shares }
SetShareFloor( 100 );
```

18.29 SetShareSize

```
SetShareSize( Shares: integer );
```

ChartScripts SimuScripts PerfScripts CMScripts

Description

Use **SetShareSize** to assign a fixed number of *Shares* (or contracts) per Position in your script. Subsequent trades will use the number of *Shares* or contracts that you specified.

Remarks

- Use of **SetShareSize** is generally discouraged since SimuScripts are available to set position sizing in all tools (see **SetPositionSizeShares**). The function is primarily for compatibility with earlier versions of Wealth-Lab.
- When using the \$imulator tool or Portfolio Simulation mode in the Position Sizing

control, you must choose the radio button for **SetShare/PositionSize Value** to enable **SetShareSize** to influence position sizing.

- In Raw Profit modes, **SetShareSize** will override the Position Sizing control's selection. *Exception:* **SetShareSize** has no effect in a Raw Profit WatchList or ChartScript Ranking.
- You can use both **SetPositionSize** and **SetShareSize** in the same script. As these functions are **global** in nature, the next time a trade is processed it will use the value from the function last called.

Example

```
{ Assign a trade size of 200 shares to new positions }
SetShareSize( 200 );
```

18.30 SetSlippage

SetSlippage(EnableSlippage: boolean; Slippage: float; LimitOrders: boolean);

ChartScripts SimuScripts PerfScripts CMScripts

Description

Lets you override the default Slippage settings in the **Options Dialog|Trading Costs/Control** from within your script.

EnableSlippage Controls whether Slippage is on (activated) or off.

Slippage Controls the amount of Slippage to use.

LimitOrders Controls whether prices must move to at least the Slippage-adjusted amount in order for Limit and Stop orders to be executed.

Remarks

- Settings for the most recent SetSlippage call are used for the trading signals that follow (see example).
- See the **Options Dialog|Trading Costs/Control** topic in the Wealth-Lab User Guide for details on how Slippage affects entry and exit price.

Example

```
var Bar: integer;
InstallTimeBasedExit( 10 );
InstallStopLoss( 5 );
for Bar := 20 to BarCount - 1 do
begin
{ Enable 2 units of slippage for the Autostops }
SetSlippage( true, 2, true );
ApplyAutoStops( Bar );

{ Disable slippage for other [manual] trading signals }
SetSlippage( false, 2, true );
// rest of trading system...
end;
```

18.31 ShortAtClose

ShortAtClose(Bar: integer; SignalName: string): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Establishes a short Position at market close of the specified *Bar*.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Entry Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.
- **ShortAtClose** will return boolean *false* if the signal fails to establish a new position. This can occur, for example, when using 100% equity position sizing without Leeway.

Note: AtClose orders can be difficult to realize in practice.

Example

```
{ Short at Close on a TD Power of 9 Signal }
var BAR: integer;
for Bar := 40 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    if CumUp( Bar, #Close, 4 ) = 9 then
      ShortAtClose( Bar, 'TD Power of 9' );
    end
  else
  begin
    { ... Exit Rules ... }
  end;
end;
```

18.32 ShortAtLimit

ShortAtLimit(Bar: integer; LimitPrice: float; SignalName: string): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Establishes a short Position if prices reach the indicated *LimitPrice*. The Position will be opened if prices meet or exceed the *LimitPrice* on the specified *Bar*. If prices open above the *LimitPrice*, the Position will be established at open price. If prices fail to reach the *LimitPrice* objective, a Position is not established and the function returns false.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Entry Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.
- For futures symbols, *LimitPrice* is automatically rounded to the nearest Tick value.

Example

```
{ Short the next bar at limit price of recent 10 bar high }
var BAR: integer;
```

```

var p: float;
for Bar := 40 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    p := Highest( Bar, #Close, 10 );
    ShortAtLimit( Bar + 1, p, '' );
  end
  else
  begin
    { ... Exit Rules ... }
  end;
end;

```

18.33 ShortAtMarket

ShortAtMarket(Bar: integer; SignalName: string): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Establishes a short Position at market open of the specified *Bar*.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- The string, which may be a blank string, passed as the *SignalName* parameter will appear in the Entry Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.
- **ShortAtMarket** will return boolean *false* if the signal fails to establish a new position. This can occur, for example, when using 100% equity position sizing without Leeway.

Example

```

{ Establish a short position if RSI gets overbought }
var BAR: integer;
for Bar := 40 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    if RSI( Bar, #Close, 20 ) > 70 then
      ShortAtMarket( Bar + 1, 'RSI Short Signal' );
    end
  else
  begin
    { ... Exit Rules ... }
  end;
end;

```

18.34 ShortAtStop

ShortAtStop(Bar: integer; StopPrice: float; SignalName: string): boolean;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Establishes a short Position if prices reach the indicated *StopPrice*. The Position will be opened if prices meet or go below the specified *StopPrice* on the specified *Bar*. If prices open below the *StopPrice*, the Position will be established at open price. If prices fail to reach the *StopPrice* objective, a Position is not established and the function returns false.

Remarks

- Slippage, when activated, can affect the trade's execution price.
- The string passed as the *SignalName* parameter, which may be a blank string, will appear in the Entry Signal column in the Trades View for ChartScript and \$imulator windows, or in the Signal Name column for the Scans tools.
- For futures symbols, *StopPrice* is automatically rounded to the nearest Tick value.

Example

```
{ Enter short if prices go a BIT lower than today's low }
var BAR: integer;
var sp: float;
for Bar := 40 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    sp := PriceLow( Bar ) * 0.95;
    ShortAtStop( Bar + 1, sp, '' );
  end
  else
  begin
  { ... Exit Rules ... }
  end;
end;
```

18.35 SplitPosition

SplitPosition(Position: integer; RetainPct: float): integer;

ChartScripts SimuScripts PerfScripts CMScripts

Description

Splits a single *Position* into two and returns the Position Number of the new position. You can then close off one of the Positions if desired. The *RetainPct* parameter specifies the percentage of shares/contracts to retain in the original position. For example, to keep 75% of the Position, specify 75 for this parameter. The new Position will contain 25% of the shares of the original Position.

Remarks

- The Position Number of the original position is not affected by **SplitPosition**. The new position is added to the **PositionCount** and becomes the **LastActivePosition**.
- When using an exact 50% split for odd-sized Positions, e.g. 7, 11, or 13 contracts, it's possible that floating point rounding errors can cause the odd lot to be exited in some cases and retained for others. You can work around this issue by passing a number slightly less than or greater than 50 for the *RetainPct* parameter. For example, use 49.999 to force the sale of the odd share (and retain the lesser amount).

- See the *SplitPosition Tutorial* for additional information and examples in the Wealth-Lab Knowledge Base.

Special Notes:

- Since **SplitPosition** has the effect of creating multiple positions on the entry bar, trading systems that use this function cannot be supported by the Order Manager.
- "Merging" positions is currently not possible.

Example

```
{ Split our Position into two and sell one half }
var BAR, NP: integer;
for Bar := 40 to BarCount - 1 do
begin
  if not LastPositionActive then
  begin
    { ... Entry Rules ... }
  end
  else
  begin
    { ... Exit Rules ... }
  end;
  { Exit half after 10 days }
  if LastPositionActive then
  begin
    if Bar - PositionEntryBar( LastPosition ) > 10 then
    begin
      np := SplitPosition( LastPosition, 50 );
      SellAtClose( Bar, np, '' );
    end;
  end;
end;
```