

# AmiBroker 5.00

## User's Guide



Copyright (C)1995–2007 AmiBroker.com.  
All rights reserved.



# Table of Contents

Contents.....	1
<b>Introduction.....</b>	<b>1</b>
About AmiBroker Editions.....	2
Quick Tour.....	6
Getting started.....	6
Hardware requirements.....	6
Supported operating systems.....	6
Installation and running.....	7
Getting help.....	9
<b>Tutorial.....</b>	<b>10</b>
Basic operations.....	10
Adding a new symbol.....	10
Removing a symbol.....	10
Splitting a stock.....	11
Deleting quotation.....	11
Adding/removing symbol from favourites.....	11
Merging quotations of two symbols.....	12
Metastock importer window.....	16
Beginners' charting guide.....	25
How to use drag-and-drop charting interface.....	33
User interface customization.....	36
Working with chart sheets and window layouts.....	39
Using layers.....	44
Using Web Research window.....	49
Using account manager.....	53
Using fundamental data.....	56
How to get quotes from various markets.....	58
How to set up AmiBroker with eSignal feed (RT version only).....	60
How to set up AmiBroker with myTrack feed (RT version only).....	63
How to use AmiBroker with external data source (Quote Tracker).....	64
How to set up AmiBroker with IQFeed feed (RT version only).....	69
How to use AmiBroker with Interactive Brokers TWS.....	75
How to use AmiBroker with external DDE data source.....	81
How to work with Real-Time data plugins.....	85
How to use AmiBroker with external data source (Quotes Plus, TC2000/TCNet/TC2005, FastTrack, Metastock).....	88
How to update US quotes automatically using AmiQuote.....	91
How to download quotes manually using AmiQuote.....	92
Understanding AmiBroker database concepts.....	92
Background.....	92
AmiBroker database structure.....	93
What about the external data?.....	94
Understanding categories.....	95
Working with sectors and industries.....	102
Working with watch lists.....	109
Understanding how AFL works.....	113
Creating your own indicators.....	117

# Table of Contents

## Tutorial

Using graph styles, colors, titles and parameters in Indicators.....	127
How to create your own exploration.....	130
How to write your own chart commentary.....	133
Using studies in AFL formulas.....	135
Back-testing your trading ideas.....	146
Portfolio-level backtesting.....	152
Reading backtest report.....	153
How to optimize trading system.....	157
Back-testing systems for futures contracts.....	160
Pyramiding (scaling in/out) and mutiple currencies in the portfolio backtester.....	165
Using formula-based alerts.....	167
Using interpretation window.....	167
Multiple Time Frame support in AFL.....	173
Video Tutorials (on-line).....	174

## AmiBroker Reference Guide.....174

Windows.....	175
Chart window pane.....	176
Parameters window.....	178
Study drawing tools.....	185
Line study properties window.....	186
Text box properties window.....	187
Formula Editor.....	192
Risk-Yield Map window.....	192
Database Settings.....	194
Intraday Settings window.....	196
Preferences window.....	207
Customize tools window.....	207
Symbol tree window.....	207
Information window.....	210
Notepad window.....	210
Quote Editor window.....	211
Symbol Finder window (F3).....	211
Financial data window.....	212
Profile view pane.....	212
Special fields encoding scheme.....	213
Assignment organizer window.....	213
Composite recalculation window.....	214
Categories window.....	215
ASCII Import Wizard.....	218
Metastock importer window.....	219
Portfolio management window.....	220
Real-time quote window.....	221
Easy alerts window.....	224
Bar Replay window.....	225
Formula Editor.....	230
Quick review window.....	231
Automatic analysis window.....	233

# Table of Contents

## AmiBroker Reference Guide

Filter settings window.....	233
System test settings window.....	239
System test report window.....	243
Commission window.....	244
Commentary window.....	244
Plugins window.....	245
Indicator Maintenance Wizard.....	246
Menus.....	247
File menu.....	249
Edit menu.....	250
View menu.....	252
Insert menu.....	256
Format menu.....	256
Symbol menu.....	258
Analysis menu.....	259
Tools menu.....	260
Window menu.....	261
Help menu.....	262
AFL Editor menu.....	264
Automatic Analysis result list context menu.....	266
Chart context menu.....	267
Layouts context menu.....	267
Formula (chart) context menu.....	268
Layers context menu.....	270
Keyboard shortcuts.....	272
Import ASCII.....	272
How does it work?.....	273
Format definition commands.....	285
Comments.....	285
Usage examples.....	287
Default behaviour.....	288
User-definable file types and formats.....	288
Ticker aliases.....	288
AmiBroker's OLE Automation Object Model.....	289
Index of objects.....	289
ADQuotation.....	289
ADQuotations.....	290
Analysis.....	291
Application.....	292
Window.....	292
Windows.....	293
Commentary.....	293
Document.....	293
Documents.....	294
Market.....	294
Markets.....	294
Quotation.....	294
Quotations.....	295

## Table of Contents

### AmiBroker Reference Guide

Stock.....	296
Stocks.....	296
Practical Examples:.....	299
AmiQuote's OLE Automation Object Model.....	299
Index of objects.....	300
Document.....	301

### Technical analysis guide.....301

Introduction.....	301
Basic tools.....	301
Price charts.....	302
Trend lines.....	303
Moving averages.....	304
Regression channels.....	305
Fibonacci Retracement.....	305
Fibonacci Time Zones.....	305
Bollinger bands.....	306
Indicators.....	306
Accumulation/Distribution.....	307
Advance–Decline line (AD–Line).....	307
ADX / Directional Movement Index.....	308
CCI – Commodity Channel Index.....	308
Chaikin Oscillator.....	309
MACD – Moving Average Convergence/Divergence.....	309
Money Flow Index.....	310
Negative Volume Index.....	310
OBV – On Balance Volume.....	311
Parabolic SAR (Stop–And–Reverse).....	311
RS – Relative Strength (comparative).....	311
RSI – Relative Strength Index.....	312
ROC – Price Rate Of Change.....	313
Stochastic Slow.....	313
TRIN – Arms Index.....	314
TRIX – TRiple eXponential.....	314
Ultimate Oscillator.....	315
VAP – Volume At Price histogram.....	315
Relative Performance chart.....	316

### AmiBroker Formula Language (AFL).....317

AFL Reference Manual.....	317
Introduction.....	317
Basics.....	317
Lexical elements.....	318
Language structure.....	331
Keywords.....	345
AFL Function Reference – Categorized list of functions.....	352
AFL Function Reference – Alphabetical list of functions.....	359
#include – preprocessor include command.....	360

## Table of Contents

### AmiBroker Formula Language (AFL)

#INCLUDE_ONCE – preprocessor include (once) command.....	361
#PRAGMA – sets AFL pre-processor option.....	362
ABS – absolute value.....	364
ACCDIST – accumulation/distribution.....	365
ACOS – arccosine function.....	366
ADDCOLUMN – add numeric exploration column.....	369
ADDTEXTCOLUMN – add text exploration column.....	370
ADDTOCOMPOSITE – add value to composite ticker.....	372
ADLINE – advance/decline line.....	373
ADVISSUES – advancing issues.....	374
ADVOLUME – advancing issues volume.....	375
ADX – average directional movement index.....	376
ALERTIF – trigger alerts.....	378
ALMOSTEQUAL – rounding error insensitive comparison.....	379
AMA – adaptive moving average.....	380
AMA2 – adaptive moving average.....	381
APPLYSTOP – apply built-in stop.....	385
ASC – get ASCII code of character.....	386
ASIN – arcsine function.....	387
ATAN – arc tan.....	388
ATAN2 – calculates arctangent of y/x.....	389
ATR – average true range.....	391
BARINDEX – get zero-based bar number.....	393
BARSSINCE – bars since.....	395
BBANDBOT – bottom bollinger band.....	396
BBANDTOP – top bollinger band.....	397
BEGINVALUE – Value of the array at the begin of the range.....	398
CATEGORYADDSYMBOL – adds a symbol to a category.....	399
CATEGORYFIND – search for category by name.....	400
CATEGORYGETNAME – get the name of a category.....	401
CATEGORYGETSYMBOLS – retrieves comma-separated list of symbols belonging to given category.....	403
CATEGORYREMOVESYMBOL – remove a symbol from a category.....	404
CCI – commodity channel index.....	406
CEIL – ceil value.....	407
CHAIKIN – chaikin oscillator.....	408
CLIPBOARDGET – retrieves current contents of Windows clipboard.....	409
CLIPBOARDSET – copies the text to the Windows clipboard.....	410
COLORHSB – specify color using Hue-Saturation-Brightness.....	412
COLORRGB – specify color using Red-Green-Blue components.....	413
CORRELATION – correlation.....	414
COS – cosine.....	415
COSH – hyperbolic cosine function.....	416
CREATEOBJECT – create COM object.....	417
CREATESTATICOBJECT – create static COM object.....	418
CROSS – crossover check.....	420
CUM – cumulative sum.....	422
DATE – date.....	424

## Table of Contents

### AmiBroker Formula Language (AFL)

DATENUM – date number.....	425
DATETIME – retrieves encoded date time.....	426
DATETIMECONVERT – date/time format conversion.....	428
DATETIMETOSTR – convert datetime to string.....	429
DAY – day of month.....	430
DAYOFWEEK – day of week.....	431
DAYOFYEAR – get ordinal number of day in a year.....	432
DECISSUES – declining issues.....	433
DECVOLUME – declining issues volume.....	434
DEMA – double exponential moving average.....	436
EMA – exponential moving average.....	439
ENABLEROTATIONALTRADING – Turns on rotational–trading mode of the backtester.....	441
ENABLESCRIPT – enable scripting engine.....	442
ENABLETEXTOUTPUT – enables/disables text output in the Chart Commentary window.....	443
ENCODECOLOR – encodes color for indicator title.....	445
ENDVALUE – value of the array at the end of the selected range.....	446
EQUITY – calculate single–symbol equity line.....	449
EXP – exponential function.....	450
EXREM – remove excessive signals.....	452
EXREMSpan – remove excessive signals spanning given number of bars.....	453
FCLOSE – close a file.....	454
FDELETE – deletes a file.....	455
FEOF – test for end–of–file.....	456
FFT – performs Fast Fourier Transform.....	459
FGETS – get a string from a file.....	460
FGETSTATUS – retrieves file status/properties.....	462
FLIP –.....	463
FLOOR – floor value.....	464
FMKDIR – creates (makes) a directory.....	465
FOPEN – open a file.....	466
FOREIGN – access foreign security data.....	469
FPUTS – write a string to a file.....	471
FRAC – fractional part.....	472
FRMDIR – removes a directory.....	473
FULLNAME – full name of the symbol.....	474
GAPDOWN – gap down.....	475
GAPUP – gap up.....	476
GETBASEINDEX – retrieves symbol of relative strength base index.....	477
GETCATEGORYSYMBOLS – retrieves comma–separated list of symbols belonging to given category.....	478
GETCHARTID – get current chart ID.....	479
GETCURSORMOUSEBUTTONS – get current state of mouse buttons.....	480
GETCURSORXPOSITION – get current X position of mouse pointer.....	481
GETCURSORYPOSITION – get current Y position of mouse pointer.....	482
GETDATABASENAME – retrieves folder name of current database.....	483
GETEXTRADATA – get extra data from external data source.....	485
GETFNData – get fundamental data.....	487
GETOPTION – gets the value of option in automatic analysis settings.....	489



## Table of Contents

### AmiBroker Formula Language (AFL)

GETPERFORMANCECOUNTER – retrieves the current value of the high-resolution performance counter.....	491
GETPLAYBACKDATETIME – get bar replay position date/time.....	492
GETPRICESTYLE – get current price chart style.....	493
GETRTDATA – retrieves the real-time data fields.....	495
GETRTDATAFOREIGN – retrieves the real-time data fields (for specified symbol).....	497
GETSCRIPTOBJECT – get access to script COM object.....	498
GETTRADINGINTERFACE – retrieves OLE automation object to automatic trading interface.....	499
GFXARC – draw an arc.....	500
GFXCHORD – draw a chord.....	501
GFXCIRCLE – draw a circle.....	502
GFXDRAWTEXT – draw a text (clipped to rectangle).....	505
GFXELLIPSE – draw an ellipse.....	506
GFXGRADIENTRECT – draw a rectangle with gradient fill.....	507
GFXLINETO – draw a line to specified point.....	508
GFXMOVETO – move graphic cursor to new position.....	509
GFXPIE – draw a pie.....	510
GFXPOLYGON – draw a polygon.....	511
GFXPOLYLINE – draw a polyline.....	512
GFXRECTANGLE – draw a rectangle.....	513
GFXROUNDRECT – draw a rectangle with rounded corners.....	514
GFXSELECTFONT – create / select graphic font.....	515
GFXSELECTPEN – create / select graphic pen.....	516
GFXSELECTSOLIDBRUSH – create / select graphic brush.....	517
GFXSETBKCOLOR – set graphic background color.....	518
GFXSETBKMODE – set graphic background mode.....	519
GFXSETOVERLAYMODE – set low-level graphic overlay mode.....	520
GFXSETPIXEL – set pixel at specified position to specified color.....	521
GFXSETTEXTALIGN – set text alignment.....	523
GFXSETTEXTCOLOR – set graphic text color.....	524
GFXTEXTOUT – writes text at the specified location.....	525
GROUPID – get group ID/name.....	526
HHV – highest high value.....	529
HHVBARS – bars since highest high.....	530
HIGHEST – highest value.....	531
HIGHESTBARS – bars since highest value.....	532
HIGHESTSINCE – highest value since condition met.....	533
HIGHESTSINCEBARS – bars since highest value since condition met.....	534
HOLD – hold the alert signal.....	535
HOURL – get current bar's hour.....	536
IIF – immediate IF function.....	541
INDUSTRYID – get industry ID / name.....	542
INSIDE – inside day.....	543
INT – integer part.....	544
INTERVAL – get bar interval (in seconds).....	545
INWATCHLIST – watch list membership test (by ordinal number).....	546
INWATCHLISTNAME – watch list membership test (by name).....	547
ISCONTINUOUS – checks 'continuous quotations' flag state.....	548

## Table of Contents

### AmiBroker Formula Language (AFL)

ISEMPTY – empty value check.....	549
ISFAVORITE – check if current symbol belongs to favorites.....	550
ISFINITE – check if value is not infinite.....	551
ISINDEX – check if current symbol is an index.....	552
ISNAN – checks for NaN (not a number).....	553
ISNULL – check for Null (empty) value.....	554
ISTRUE – true value (non-empty and non-zero) check.....	555
LASTVALUE – last value of the array.....	558
LINEARRAY – generate trend-line array.....	559
LINEARREG – linear regression end-point.....	560
LINREGINTERCEPT –.....	561
LINREGSLOPE – linear regression slope.....	562
LLV – lowest low value.....	565
LLVBARS – bars since lowest low.....	566
LOG – natural logarithm.....	567
LOG10 – decimal logarithm.....	568
LOWEST – lowest value.....	569
LOWESTBARS – bars since lowest.....	570
LOWESTSINCE – lowest value since condition met.....	571
LOWESTSINCEBARS – bars since lowest value since condition met.....	572
MA – simple moving average.....	577
MACD – moving average convergence/divergence.....	578
MARKETID – market ID / name.....	579
MAX – maximum value of two numbers / arrays.....	581
MDI – minus directional movement indicator (–DI).....	582
MEDIAN – calculate median (middle element).....	583
MFI – money flow index.....	584
MIN – minimum value of two numbers / arrays.....	586
MINUTE – get current bar's minute.....	587
MONTH – month.....	588
MTRANDOM – Mersene Twister random number generator.....	589
NAME – ticker symbol.....	592
NOTEGET – retrieves the text of the note.....	593
NOTESSET – sets text of the note.....	594
NOW – gets current system date/time.....	595
NUMTOSTR – convert number to string.....	597
NVI – negative volume index.....	598
NZ – Null (Null/Nan/Infinity) to zero.....	599
OBV – on balance volume.....	600
OPTIMIZE – define optimization variable.....	603
OSCP – price oscillator.....	604
OSCV – volume oscillator.....	605
OUTSIDE – outside bar.....	606
PARAM – add user user–definable numeric parameter.....	610
PARAMCOLOR – add user user–definable color parameter.....	612
PARAMDATE – add user user–definable date parameter.....	613
PARAMFIELD – creates price field parameter.....	614
PARAMLIST – creates the parameter that consist of the list of choices.....	615

## Table of Contents

### AmiBroker Formula Language (AFL)

PARAMSTR – add user user–definable string parameter.....	616
PARAMSTYLE – select styles applied to the plot.....	618
PARAMTIME – add user user–definable time parameter.....	619
PARAMTOGGLE – create Yes/No parameter.....	620
PARAMTRIGGER – creates a trigger (button) in the parameter dialog.....	621
PDI – plus directional movement indicator.....	622
PEAK – peak.....	623
PEAKBARS – bars since peak.....	624
PERCENTILE – calculate percentile.....	625
PLOT – plot indicator graph.....	630
PLOTFOREIGN – plot foreign security data.....	631
PLOTGRID – Plot horizontal grid line.....	632
PLOTOHLC – plot custom OHLC chart.....	633
PLOTSHAPES – plots arrows and other shapes.....	636
PLOTTEXT – write text on the chart.....	637
PLOTVAPOVERLAY – plot Volume–At–Price overlay chart.....	638
POPUPWINDOW – display pop–up window.....	639
PREC – adjust number of decimal points of floating point number.....	640
PREFS – retrieve preferences settings.....	642
PRINTF – Print formatted output to the output window.....	643
PVI – positive volume index.....	644
RANDOM – random number.....	646
REF – reference past/future values of the array.....	652
RELSTRENGTH – comparative relative strength.....	653
REQUESTTIMEDREFRESH – forces periodical refresh of indicator pane.....	654
RESTOREPRICEARRAYS – restore price arrays to original symbol.....	655
RMI – Relative Momentum Index.....	656
ROC – percentage rate of change.....	658
ROUND – round number to nearest integer.....	659
RSI – relative strength index.....	661
RWI – random walk index.....	662
RWIHI – random walk index of highs.....	663
RWILO – random walk index of lows.....	664
SAR – parabolic stop–and–reverse.....	665
SAY – speaks provided text.....	667
SECOND – get current bar's second.....	668
SECTORID – get sector ID / name.....	669
SELECTEDVALUE – retrieves value of the array at currently selected date/time point.....	671
SETBACKTESTMODE – Sets working mode of the backtester.....	672
SETBARSREQUIRED – set number of previous and future bars needed for script/DLL to properly execute.....	674
SETCHARTBKCOLOR – set background color of a chart.....	675
SETCHARTBKGRADIENTFILL – enables background gradient color fill in indicators.....	676
SETCHARTOPTIONS – set/clear/overwrite defaults for chart pane options.....	678
SETCUSTOMBACKTESTPROC – define custom backtest procedure formula file.....	679
SETFOREIGN – replace current price arrays with those of foreign security.....	681
SETFORMULANAME – set the name of the formula.....	682
SETOPTION – sets options in automatic analysis settings.....	685

## Table of Contents

### AmiBroker Formula Language (AFL)

SETPOSITIONSIZE – set trade size.....	688
SETSORTCOLUMNS – sets the columns which will be used for sorting in AA window.....	689
SETTRADEDELAYS – allows to control trade delays applied by the backtester.....	690
SIGN – returns the sign of the number/array.....	691
SIGNAL – macd signal line.....	692
SIN – sine function.....	693
SINH – hyperbolic sine function.....	694
SQRT – square root.....	695
STATICVARGET – gets the value of static variable.....	696
STATICVARGETTEXT – gets the value of static variable as string.....	697
STATICVARREMOVE – remove static variable.....	698
STATICVARSET – sets the value of static variable.....	699
STATICVARSETTEXT – Sets the value of static string variable.....	700
STATUS – get run-time AFL status information.....	702
STDERR – standard error.....	703
STDEV – standard deviation.....	705
STOCHD – stochastic slow %D.....	706
STOCHK – stochastic slow %K.....	707
STREXTRACT – extracts given item (substring) from comma-separated string.....	708
STRFIND – find substring in a string.....	709
STRFORMAT – Write formatted output to the string.....	711
STRLEFT – extracts the leftmost part.....	712
STRLEN – string length.....	713
STRMID – extracts part of the string.....	714
STRREPLACE – string replace.....	715
STRRIGHT – extracts the rightmost part of the string.....	716
STRTODATETIME – convert string to datetime.....	717
STRTOLOWER – convert to lowercase.....	718
STRTONUM – convert string to number.....	719
STRTOUPPER – convert to uppercase.....	720
STUDY – reference hand-drawn study.....	721
SUM – sum data over specified number of bars.....	724
TAN – tangent function.....	725
TANH – hyperbolic tangent function.....	726
TEMA – triple exponential moving average.....	727
TIMEFRAMECOMPRESS – compress single array to given time frame.....	729
TIMEFRAMEEXPAND – expand time frame compressed array.....	731
TIMEFRAMEGETPRICE – retrieve O, H, L, C, V values from other time frame.....	733
TIMEFRAMEMODE – switch time frame compression mode.....	735
TIMEFRAMERESTORE – restores price arrays to original time frame.....	737
TIMEFRAMESET – switch price arrays to a different time frame.....	740
TIMENUM – get current bar time.....	741
TRIN – traders (Arms) index.....	742
TRIX – triple exponential smoothed price.....	743
TROUGH – trough.....	744
TROUGHBARS – bars since trough.....	745
TSF – time series forecast.....	746
ULTIMATE – ultimate oscillator.....	747

# Table of Contents

## AmiBroker Formula Language (AFL)

UNCISSUES – unchanged issues.....	748
UNCVOLUME – unchanged issues volume.....	749
VALUEWHEN – get value of the array when condition met.....	751
VARGET – gets the value of dynamic variable.....	752
VARGETTEXT – gets the text value of dynamic variable.....	753
VARSET – sets the value of dynamic variable.....	754
VARSETTEXT – sets dynamic variable of string type.....	755
VERSION – get version info.....	756
WILDERS – Wilder's smoothing.....	757
WMA – weighted moving average.....	758
WRITEIF – commentary conditional text output.....	760
WRITEVAL – write number or value of the array.....	764
YEAR – year.....	765
ZIG – zig-zag indicator.....	766
_DEFAULT_NAME – retrieve default name of the plot.....	767
_N – no text output.....	772
_PARAM_VALUES – retrieve param values string.....	773
_SECTION_BEGIN – section begin marker.....	775
_SECTION_END – section end marker.....	777
_SECTION_NAME – retrieve current section name.....	778
_TRACE – print text to system debug viewer.....	778
AFL Error List.....	793
Calculating multiple-security statistics with AddToComposite function.....	797
Equity function, Individual and Portfolio Equity Charts.....	801
Functions accepting variable periods.....	802
User-definable functions, procedures. Local/global scope.....	804
AFL Tools.....	804
Automatic technical analysis.....	805
Automatic analysis window.....	806
Formula Editor.....	806
Guru Advisor Commentary window.....	806
AFL Scripting Host.....	806
Basics.....	806
Requirements.....	807
Enabling AFL Scripting Host.....	811
Further information.....	811
Component Object Model support in AFL.....	811
Introduction.....	812
Calling functions defined in script.....	813
Using external COM/ActiveX objects in AFL.....	815
Plug-in in AFL.....	816
Common Coding mistakes in AFL.....	820
Portfolio Backtester Interface Reference Guide.....	832
How to add user-defined metrics to backtest/optimization report.....	840
Using low-level graphics functions.....	840
Usage examples:.....	845

## Table of Contents

<b>What's new in latest version?</b> .....	<b>875</b>
Usage notes on new features in 5.00.....	876
Usage notes on new features in 4.90.....	877
Usage notes on new features in 4.80.....	877
Usage notes on new features in 4.70.....	879
Usage notes on new features in 4.60.....	880
Usage notes on new features in 4.50.....	882
<b>Technical information</b> .....	<b>882</b>
Troubleshooting guide.....	885
Files and directories used by AmiBroker.....	886
Crash recovery system and automatic bug reporting.....	890
<b>How to purchase AmiBroker ?</b> .....	<b>892</b>
<b>Credits</b> .....	<b>title</b>

**Thank you for choosing AmiBroker.** This guide will help you get up and running.

AmiBroker is a comprehensive technical analysis program, with an advanced charting, back-testing and scanning capabilities. It gives everything you need to trade successfully. Just check out our [quick features tour](#) to find out what is included in this powerful software package.

If you are a **first time user** and just installed the software please check out [Tutorial section](#) that will guide you through most important aspects of using AmiBroker.

The next chapter – [Reference guide](#) – provides detailed description of every window and more technical documentation covering ASCII importer and automation interface.

In the [Technical analysis guide](#) you will find material that will introduce you to the world of charting and technical indicators.

The next part of the guide describes [AmiBroker Formula Language](#) – a powerful tool that allows you to create your own trading systems, scans, custom indicators and commentaries. You will find the description of the language and its syntax, a complete reference of all functions and more.

The last part is provided for the user's of previous versions – this chapter will help them finding out what new features were added without the need to re-read all documentation.

## About AmiBroker Editions

AmiBroker software is currently available in 2 editions: Standard and Professional.

The following table summarizes differences between these two editions:

Feature	Standard Edition	Professional Edition
End-of-day charting/backtesting/scanning	Yes	Yes
1-, 5-, 15- minute, hourly Intraday charting/backtesting/scanning	Yes	Yes
Custom minute bars	Yes	Yes
Tick charts/backtesting/scanning	No	<b>Yes*</b>
5-second, 15-second bar charts/backtesting/scanning	No	<b>Yes</b>
Streaming real time quote display	10 symbols	<b>UNLIMITED symbols</b>
Time and Sales window	1 symbol	<b>UNLIMITED symbols</b>
GetRTData / GetRTDataForeign AFL function	No	<b>Yes</b>
Wait for backfill in Automatic Analysis	No	<b>Yes</b>
Automatically updating real time charts	Yes	<b>Yes</b>
Maximum Adverse/Favourable Excursion Distribution charts in Portfolio backtest reports	No	<b>Yes</b>
64-bit version	No	<b>Yes</b>

Requires RT data subscription	No	Not required, but nice to have  (Professional Edition can work with EOD data too, but real-time features (like real-time quote) of course are require real-time data source)
-------------------------------	----	--

\* – this feature is available only using eSignal RT, Interactive Brokers, DDE feed

In the future the Professional Edition may have additional extra features not available in Standard Edition. For pricing and ordering information check out [How to order](#) section.

## Quick Tour

### Basic features

#### Powerful charting

- **object-oriented drawing tools** (*trend lines, rays, parallel lines, regression channels, fibonacci retracement, expansion, Fibonacci time extensions, Fibonacci timezone, arc, gann square, gann square, cycles, circles, rectangles, text on the chart, and more*)
- **drag-and-drop indicator creation** – allows you to create complex indicators without writing single line of code
- **modern, fully customizable user interface**
- instant viewing of intraday/daily/weekly/monthly charts in line, bar or candlestick styles overlaid with configurable moving averages, Bollinger bands, Volume chart, SAR, etc.
- ability to display most common 1-, 5-, 15-, 60- minute intraday charts as well as **fully customizable N-minute charts (where N is 1..1380 )**
- **5-second and 15-second bar charts (RT version)**
- **tick charts, custom N-tick charts (RT version)**
- **multiple time frame charts**
- on-the-fly time compression – no need to wait when switching between various chart periodicities
- **relative performance charts**
- tens of most popular indicators built-in including ROC, RSI, MACD, OBV, CCI, MFI, NVI, Stochastics, Ultimate oscillator, DMI, ADX, Parabolic SAR, TRIN, Advance/Decline line, Accumulation/Distribution, TRIX, Chaikin oscillator, unique risk-to-yield map and more
- study drawing tools including trend lines, horizontal/vertical lines, Fibonacci retracements and timezones, text boxes
- multiple chart panes, windows, different views and time scales are possible all at the same time
- extremely fast zooming and live scrolling

#### Multiple data feeds

AmiBroker is capable of handling virtually ANY exchange in the world.



- **Real-time streaming quotes via eSignal's TurboFeed featuring access to all US exchanges and major European exchanges.**
- **Real-time streaming quotes via myTRACK feed, IQFeed, QCharts/Quote.com, QuoteTracker, Marketcast, Interactive Brokers, any DDE-enabled data feed**
- **Direct feed from Quotes Plus, TC2000, FastTrack and Metastock (including intraday) databases.**  
[Read more...](#)
- User-configurable ASCII import wizard – allows you to read quotes in the format you can define (including intraday)!
- Built-in Metastock(R) database importer – reads directly all symbols from your Metastock database (works with both EOD and intraday modes) in a matter of seconds!
- AmiQuote downloader program provides quick way of obtaining free end-of-day from major world exchanges (all US markets, LSE, ASX, Paris, Milan, Frankfurt)
- Free FOREX data downloadable via AmiQuote
- Free historical intraday delayed quotes from US exchanges downloadable via AmiQuote
- Script-driven, one-click automatic downloaders available for NYSE, Amex, Nasdaq, Australian Stock Exchange, Johannesburg Stock Exchange, Warsaw Stock Exchange

AmiBroker is successfully used in the following countries: USA, Canada, United Kingdom, Australia, Germany, Italy, Southern Africa, Poland, Holand, Norway, France, ...

For more information on data sources for AmiBroker [click here](#).

## Symbol & quotes database

AmiBroker features advanced database system that offers the following:

- **build-up and store historical tick or 5- or 15-second bar data for backtesting purposes (certain RT data sources only)**
- **build-up and store intraday minute-bar or end-of-day data for backtesting purposes**
- unlimited number of symbols and unlimited number of quotes
- multiple database support
- stores quotes, company information, financial results, categories, industry/sector information
- powerful filtering by sector, industry, group and market
- innovative symbol tree browser showing symbols grouped by sectors, industries, indexes
- automatic handling for composites (number and volumes of advancing, declining and unchanged symbols)
- automation support allowing you to control your database from external programs written in any language including Java Script, VBScript

## AmiBroker Formula Language

### The language

The AFL is an advanced formula language that allows you to create your own indicators, trading systems and commentaries. It is specially designed for traders so writing analysis formulas is easier and quicker than in general-purpose languages.

AFL features more than 200 built-in AFL functions to use as a building blocks for your formulas. AFL includes trigonometric, averaging, statistical, data manipulation, conditional, pattern-detection and predefined indicator functions.

AFL supports unlimited variables, unlimited parentheses nesting, unlimited nested function calls and multiple

logical operators. Version 4.40 brings completely rewritten engine with native flow-control and looping (if-else, while), user-defined functions and procedures with local and global variable scope.

New version 4.50 provides native multiple time-frame support, so you can mix different bar intervals in single formula.

### Formula Editor / Drag-drop charting

[Formula Editor](#) allows you to quickly re-create any indicator/study found in the literature. [Drag and drop charting](#) allows to create complex overlays, indicators-on-indicators and more. Among other things it is possible to:

- any number of graphs that can be overlaid in the same chart pane
- modify built-in indicators
- custom or automatic scaling
- flexible grids
- access to composite data (number/volume of advancing, declining, unchanged issues)

### Formula – based alerts

- Ability to write complex formula-based alerts that can be displayed on the screen, sent to you via e-mail, plus play a user-defined WAV file.
- Ability to run external applications via alerts – this allows automated trade execution

### PORTFOLIO-LEVEL system back-testing, optimization, explorations and screening

Screening: Automatic analysis window enables you to scan your database for symbols matching your defined buy/sell rules. AmiBroker automatically produces the report telling you if buy/sell signals occurred on given symbol in the specified period of time.

Exploration: search your database for symbols matching your criteria and create the report showing the data you want to see: indicator values, past performance, etc. Then sort the results by any value listed.

Back-testing: AmiBroker can also perform full-featured back-testing of your trading strategy, giving you an idea about performance of your system.

The back-testing engine highlights:

- **PORTFOLIO LEVEL BACKTESTING/OPTIMIZATION**
- **Three-dimensional (3D), fully animated charts of optimization results**
- **[Advanced custom backtester interface](#)**
- **[User-definable backtest metrics](#)**
- **Different position sizing / money management techniques based on Portfolio Equity**
- **Hyper-fast execution – AmiBroker can backtest 10000 symbols (3000 data bars each) = 30 million data points in FIVE minutes!**
- **Integrated support for MULTIPLE time-frames in single formula**
- **NEW Report Explorer provides great way to organize/compare/view all backtest results**
- **Scanning/Exploration/Backtest/Optimization on Real Time data (tick and up) (RT version only)**
- **Scanning/Exploration/Backtest/Optimization on intraday data (1-min bars and up)**
- **Back testing whole exchange or only limited, user-definable set matching your market, group, industry, sector selection**
- **Equity curve plotting, Equity rainbows, composite equities curves**

- Test long, short or both long and short trades
- Maximum–loss stop, profit–target stop, trailing–stop, N–bar (time) stop
- Realistic back–testing
- Ability to control position size from your formula ([Read more...](#))
- Create your own composites and scan/backtest them
- Detailed reporting giving you important statistics of your system.

Optimization: AmiBroker allows you to optimize your trading system with up to 10 optimization variables on single or **MULTIPLE securities** at once!

### Automatic Chart Commentaries and Interpretation

- Full, textual descriptions of actual situation on the market
- automatic buy–sell arrows visible on the charts
- automatic textual interpretation of indicators and price chart (View–>Interpretation)

### Scripting/COM/DLL support

- AFL engine allows embedding VBScript/JScript code within AFL formulas providing UNLIMITED possibilities
- ability to call external COM (ActiveX) objects from the AFL formula
- free SDK (software development kit) for registered users allowing writing indicator DLLs (plug–ins)
- many already available 3rd party plug–ins

### *Additional features*

#### Portfolio manager

Built–in portfolio manager helps you track your investments. It allows you to register buy/sell transactions, calculates brokerage commission, dividend (with settable dividend tax), cash deposits/withdrawals. You get the instant calculation of your equity value, percentage and point yield.

#### Scripting support

AmiBroker features automation interface that exposes objects and methods that could be accessed from any programming language including scripting dialects such as JScript (JavaScript) and VBScript. The scripting capabilities of AmiBroker allow you to automate time consuming database management tasks. Using scripting you will be able to create automatic downloaders, maintenance tools, exporters customized to your specific needs.

#### Internet integration

AmiBroker features built–in web browser that allows you to quickly view company profiles. The profile viewer is completely configurable so you can set it up for your particular exchange. The settings are market based so you can access different web sites for each market automatically. No longer will you be forced to waste your time browsing manually to get the latest news and symbol related information.

#### Configurability

AmiBroker is designed to be configurable and customizable in almost every area. It is not tied

to particular exchange or data provider. Thanks to flexible import methods and scripting you will be able to adopt it easily to your favourite market(s). Also technical analysis tools built in into AmiBroker allow you to change every parameter with easy, and if you want even more, you can create your own indicators using flexible formula language.

## Getting started

*Hardware requirements*

*Supported operating systems*

*Installation and running*

*Getting help*

## Hardware requirements

To run AmiBroker you need PC-Compatible computer meeting following minimum requirements

Standard version:

- Pentium 166 MHz or higher
- 32 MB RAM
- 8 MB hard disk space
- 256 color graphics card (high color recommended)

Real-time version:

- Pentium 450 MHz or higher
- 128 MB RAM
- 20 MB hard disk space
- 256 color graphics card (high color recommended)

## Supported operating systems

AmiBroker works on the following operating systems:

- Windows 95 + Internet Explorer 4.0 or higher installed
- Windows 95 OSR 2 + Internet Explorer 4.0 or higher installed
- Windows 98
- Windows 98SE
- Windows Millenium
- Windows NT 4.0 SP 3 (or higher) + Internet Explorer 4.0 or higher installed
- Windows 2000 (any edition)
- Windows XP (any edition)

## Installation and running

Install AmiBroker using it's setup program – it is available for download from <http://www.amibroker.com/download.html>. After downloading double click on the program's icon. This will launch the setup program – you can safely accept all default values by clicking "Next" on each page and "Install" on the last page. By default AmiBroker is installed to "C:\Program Files\AmiBroker" directory and this location is referred to as "main AmiBroker directory".

If setup program asks you to restart machine please do so to allow to replace system components.

After installation, you can start AmiBroker from Windows' standard Start->Programs->AmiBroker->AmiBroker menu.

Just after starting AmiBroker splash window shows up, then for few seconds AmiBroker loads its quotation database. Next the main AmiBroker screen appears.

AmiBroker main screen with price chart,  
MACD and RSI indicators and profile view open. (Windows version)

In default setup you can see the toolbar, workspace window with symbol list on the left side and chart windows on the right side.

The toolbar provides fast access to the most often used program functions. With the symbol list view you can select active symbol. Changing the selection will cause chart redraw and update in some information windows if they are open. The chart windows let you to analyse current price trends and the behaviour of technical indicators.

You can quit AmiBroker using the *File/Exit* menu item.

## Getting help

AmiBroker 4.70 features new **context-sensitive help system**, available by pressing **F1** key anywhere in the program.

When you press **F1** key while any window and any menu is shown, AmiBroker opens up a relevant help file page describing the window or menu in question. No more searching through the help file.

In addition to using F1 context-sensitive help it is **highly recommended** to read ALL [Tutorial](#) articles first. The answers to most common problems are given there. In case of major problem check [Troubleshooting guide](#). Also there is a "Search" tab on the left of this on-line help window that allows to quickly locate information by keyword(s). Just type word(s) you are looking for and click "Display".

In case of further questions/problems you may check the following resources:

- [AmiBroker web page](#) – which is searchable using "Search" box in the top left corner of the page. The page gives you an access to:
  - ◆ [AmiBroker Tips newsletter](#) containing valuable step-by-step instructions on using various aspects of AmiBroker
  - ◆ [Support area](#) – featuring additional documentation
  - ◆ [Frequently Asked Questions](#) – the list of most commonly asked questions with the answers
  - ◆ [AFL Library](#) – featuring ready-to-use AFL formulas for custom indicators, commentaries and trading systems
  - ◆ [Members area](#) – featuring material accessible by registered users only
- [AmiBroker mailing list](#) – the place where you can meet other AmiBroker users, ask questions and share with ideas (with searchable archive). To subscribe please send an empty e-mail to: [amibroker-subscribe@egroups.com](mailto:amibroker-subscribe@egroups.com). To unsubscribe please send an empty e-mail to [amibroker-unsubscribe@egroups.com](mailto:amibroker-unsubscribe@egroups.com).

Checking these places first will help me focusing on developing new features in AmiBroker. In case of

problems not covered in above resources please don't hesitate to contact me at: [support@amibroker.com](mailto:support@amibroker.com).

# Tutorial

This chapter will guide you through the most important parts of AmiBroker.

Basic tasks:

- [Basic operations](#)

User interface topics:

- [Beginners' charting guide](#)
- [How to use drag-and-drop charting interface](#)
- [Customizing user interface](#)
- [Working with chart sheets and window layouts](#)
- [Working with layers](#)
- [Using Web Research](#)
- [Using Account Manager](#)
- [Using Fundamental data](#)

Updating quotes:

- [How to get quotes from various exchanges](#)
- [Setting up eSignal RT feed \(RT version only\)](#)
- [Setting up myTrack RT feed \(RT version only\)](#)
- [Setting up Quote Tracker as a RT data source](#)
- [Setting up IQFeed RT feed \(RT version only\)](#)
- [How to use AmiBroker in Real Time mode \(RT version only\)](#)
- [Using AmiBroker with other external data source \(Quotes Plus, TC2000 / TCNet, Metastock, FastTrack\)](#)
- [Automatic update of EOD quotes for US & Canada markets from Yahoo](#)
- [Using manual mode of AmiQuote downloader \(Yahoo, MSN Money Central, Quote.com Livecharts\)](#)
- [Using Metastock importer](#)

Database management:

- [Understanding database concepts](#)
- [Understanding categories](#)
- [Working with watch lists](#)

AmiBroker Formula Language topics:

- [Understanding how AFL language works](#)
- [Creating your own indicators](#)
- [Using graph styles and colors in the indicators](#)
- [How to create your own exploration](#)
- [How to write your own chart commentary](#)
- [Using studies in your AFL formulas](#)
- [Backtesting your trading ideas](#)
- [Portfolio backtesting](#)
- [Reading backtest report](#)
- [How to optimize a trading system \(advanced\)](#)

- [Backtesting futures](#) (advanced)
- [Pyramiding/scaling and multiple currencies in the portfolio backtester](#) (advanced)
- [Using formula-based alerts](#) (advanced)
- [Using interpretation window](#) (advanced)
- [Multiple time frame support](#) (advanced)

More information:

- [Video Tutorials On-Line](#)

## Basic operations

### Adding a new symbol

In order to add a new symbol into database you can use *Symbol->New* menu item or Add symbol toolbar button.

After selecting this function you will be prompted for new ticker symbol. Please try not to exceed 26 chars. For proper import functioning you should enter the symbol with CAPITALS.

### Removing a symbol

In order to remove existing symbol from the database you can use *Symbol->Remove* menu item or Remove symbol toolbar button. After choosing this function you will be asked for confirmation of symbol removing. Note well that this operation can not be undone !!!

Removing multiple symbols at once is possible using [Assignment organizer](#).

### Splitting a stock

To perform stock split use *Symbol->Split* menu item or Split toolbar button.

AmiBroker provides easy way of handling stock splits. Program will try to guess split date and ratio by analyzing quotations. If there is just a single quotation after split this should work, if not you will be asked for split date and ratio. Note well that this operation can not be undone!!!

From version 2.0 and up the split function offers more functionality: you can use old-style ratio or you can specify a split using following expression:

$x \rightarrow y$

which means that x shares before split become y after it. For example  $2 \rightarrow 3$  means that 2 shares become 3 after the split. So ordinary split into five pieces will be  $1 \rightarrow 5$ .

As you have probably guessed it is possible now to perform reverse-split, for example  $2 \rightarrow 1$ , which means that 2 shares are joined together into 1 share.



## Deleting quotation

To delete a quotation simply select the quote you want to delete by clicking on the chart (a vertical line will appear showing selected date and quote). Then choose *Edit->Delete quotation* menu option.

To delete quotations of all stocks from given day you should use *Edit->Delete session*.

You can also use [Quote Editor](#) to delete quotes.

## Adding/removing symbol from favourites

To add the symbol to the favourites you should check favourite box in the [Information](#) window. To remove it from favourites simply uncheck that box. Alternatively you can click on the tree with the right mouse button and select "Add to favourites" and "Remove from favourites" options from the context menu.

## Merging quotations of two symbols

It happens sometimes that the ticker for the symbol is changed then you may get two tickers in your database – one holding historical quotes and the second one holding newest quotes (after name change). In order to put all quotes to the single ticker you should use *Symbol->Merge* feature. You should just select the new ticker (after name change) and choose *Symbol->Merge*. Then from the combo you should choose original ticker ("merge with") and optionally check the following fields:

- overwrite duplicate quotes – checking this option will overwrite the quotes already existing in "new" ticker with those present in "old" ticker (this should really not be the case, but may happen).
- delete "merge with" afterwards – checking this option will delete the "old" ticker after merging
- assign alias name – checking this option will copy the "old" ticker to the alias field of the "new" ticker

## Beginners' charting guide

### Introduction

AmiBroker 4.20 brings new charting engine with object-oriented manipulation of all drawings. Now you can simply move, resize, cut, copy, paste and delete all drawing objects with ease. This chapter will guide you through most important aspects of using charting tools.

Let's now take a look at the user interface:



As you can see in the center we have chart area in which price chart with moving average and Bollinger bands is plotted (**you can control the appearance of built-in charts** from [Tools->Preferences](#) window).

In the bottom of the chart you can see date axis (marked with red color), and below scroll bar and chart sheets tab control. Scroll bar can be used to display past quotes, while sheet tab allows to view different chart pages/sheets ([click here to learn more about chart sheets](#)).

To the right you can see Y-axis area (marked with blue color) that shows Y-scale and value labels. Value labels are color fields that display precisely the "last value" of plots. "Last value" is the value of the indicator (or price) for the last currently displayed (rightmost) bar. Y-axis area is used also to move/size chart vertically.

Next to the right is a drawing objects toolbar that allows you to choose from available drawing types (note that

only most popular tools are shown here, complete set is available from **Insert** menu). A special tool called "Select" (red arrow) is used to select/move/resize already drawn objects and to select quotes from the chart.

In the upper part you can see formatting toolbar that allows you to quickly modify color, style (thick/dotted) and mode (snap to price) of currently selected drawing object.

In the picture you can also see the trend line drawn with sizing handles marked. These handles are used to drag/size the object as will be explained below.

## Basic operations

### *Scrolling*

To **scroll** the chart forward/backward just drag scroll bar thumb or use < and > arrows on the left and right sides of the scroll bar. Note that using < > scroll bar arrows allows you to move chart by one bar. To scroll the chart you can also use the mouse equipped with a wheel. Just roll the wheel up and down to scroll back and forward.

### *Zooming*

To **zoom** the chart (increase or decrease number of data points (bars) displayed) you can use either **View->Zoom** menu, zoom toolbar or mouse wheel. There are following options available: zoom-in – decreases the number of data points displayed, zoom-out – increases the number of data points displayed, zoom-all – displays all available bars, zoom-normal resets number of bars displayed to the value defined in **Tools->Preferences->Charting**. Zoom-in and zoom-out options are accessible directly from the View toolbar. (see picture below). To zoom using mouse wheel just press and hold down CTRL key and roll the wheel. You can also zoom to any from-to range selected on the chart (see 'Marking range' later in this tutorial)

### *Shrinking, expanding and moving Y-axis scale*

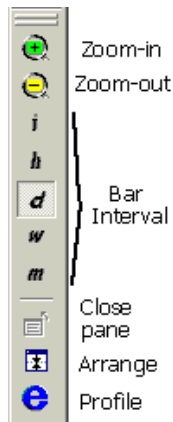
To **move** Y-axis scale hover the mouse to Y-axis area (marked with blue color in the picture above) and you will see that cursor changes to up/down arrow. Now you click and drag up/down Y axis and release button when the axis is in the correct position.

To **shrink/expand** Y-axis scale: press down SHIFT key and click in the Y-axis area, now shrink/expand Y axis scale by moving your mouse up and down. Release the button to finish.

To **reset** Y-axis scale and position simply double click in the Y-axis area.

### *Changing bar interval (periodicity)*

You can easily switch between daily/weekly/monthly and intraday intervals by choosing it from **View** menu and pressing the toolbar button (see below).



The toolbar uses following notation for intervals – **i** –intraday, **h** – hourly, **d** – daily, **w** – weekly, **m** – monthly. The **i** represents "base" intraday interval as defined in **File-->Database Settings**. Remaining intraday intervals are available from **View-->Intraday** menu.

The interval setting affects active window only, so each window can have different interval.

Please note that intraday intervals are disabled if your database is in end-of-day mode. Intraday modes are available only for databases that have "Base time interval" in **File-->Database Settings** set to anything less than end-of-day. If you for example set "Base time interval" in **File-->Database Settings** to 5-minute, all chart periodicities from 5-minutes up will be enabled.

The following intervals are built-in:

- daily
- weekly
- monthly
- hourly (intraday)
- 15-minute (intraday)
- 5-minute (intraday)
- 1-minute (intraday)
- 15-second (intraday RT only)
- 5-second (intraday RT only)
- tick (intraday RT only)

In addition to that you can define 5 custom n-minute bar intervals and 5 custom n-tick intervals in **Tools-->Preferences-->Intraday**. Custom intervals are available from **View-->Intraday** menu only.

### Selecting a quote

You can very easily see the past quote and values of indicators by using "select" mode. To **select** past quote first switch to "Select" mode (red arrow in the toolbar) then click in the chart area (but not on the drawing object). A vertical line will show up marking the quote under the cursor. The chart title will display this bar quote. Indicator panes will show indicator value for given bar. Once quote is selected you can move to previous/next quote using keyboard left and right arrow (cursor) <- and -> keys.

To **switch off** quote selection either click again on the line or click in the date axis area (marked with red color in the picture above) or click in the right margin (blank quotes) area. When selection is off chart title displays the values for last visible bar.

### Marking range

To show range marker just double click the chart at the beginning of the range and double click again at the end of the range. You can also use F12 key in conjunction with "select" mode (described above). Just select quote and press F12 for begin and SHIFT+F12 for the range end. You can switch off the range marker by pressing CTRL+F12 key or double clicking in the same place twice.

Range markers can be used to select zoom-in range (View-->Zoom-->Range) and to perform calculations on selected values via BeginValue and EndValue AFL functions.
























*Adding / closing chart panes*

Each window can consist of several panes displaying various charts / indicators. To display a new indicator in a separate chart pane just find the indicator in the Charts list (use **View -> Charts** menu) and **double-click** on the indicator name. For more information see [Drag&drop charting](#) tutorial.

To close any chart pane: click on the pane, then use either **View->Pane->Close** from main menu or **click on the pane with right mouse button** and choose **Close** from context menu.

*Using drawing tools*

AmiBroker features extensive set of drawing tools:

	Select	
	Trendline	• trend line
	Ray	• ray (new in 4.20)
	Extended	• extended line (new in 4.20)
	Vertical	• vertical line
	Horizontal	• horizontal line
	Parallel	• parallel lines (new in 4.20)
	Regression Channel	• Regression channels: Raff, standard deviation, standard error (all new in 4.20)
	Rectangle	• Fibonacci Retracement study (enhanced in 4.20)
	Ellipse	• Fibonacci Time zones study
	Text	• Fibonacci Fan
	Triangle	• Fibonacci arc
	Pitchfork	• Gann Square (new in 4.20)
	Arc	• Gann Fan (new in 4.20)
	Cycles	• Ellipse tool
	Fibonacci Retracement	• Arc tool
	Time zones	• Rectangle
	Fib.Fan	• text box tool
	Fib. arc	
	Gann square	
	Gann fan	
	Fibonacci extension	
	Time extension	

The following tools are available:

They are available from **Insert** menu and **Draw** toolbar. Each drawing object can be moved, resized, copied, deleted and modified after it is drawn.

To **draw** an object on the chart switch on appropriate tool button (see picture below) and start drawing on the chart by pointing the mouse and pressing left mouse button where you want to start the drawing. Then move the mouse. Study tracking line will appear. Release left mouse button when you want to finish drawing. You can also cancel study drawing by pressing ESC (escape) key.

If you hover your mouse over the object you will see that cursor shape changes in the proximity of the object. This means that

If cursor is near either end of the object it will change its shape to **sizing** pointer:



If the cursor is near remaining parts of the object it will change its shape to **moving** pointer:



Once object is drawn it can be selected, moved, resized, deleted, copied.



To **select** the object simply move the mouse over the object so "moving pointer" appears and click once – the object will be marked so the sizing handles (see first picture) will appear.

To **de-select** click in the blank chart space.

To **size** the object click on the sizing handle and drag to the desired location as shown in the picture.

To **move** the object click on any other part of the object and move to the desired location.

To **delete** object – select it first and press **DEL (DELETE)** key on the keyboard or use **Edit->Delete** menu or use Delete toolbar button.

To **copy** the object to the clipboard – select it first and press **Ctrl+C** or use **Edit->Copy** menu or use Copy toolbar button.

To **cut** the object – select it first and press **Ctrl+X** or use **Edit->Cut** menu or use Cut toolbar button.

To **paste** the object from the clipboard press **Ctrl+V** or use **Edit->Paste** or use Paste toolbar button. Pasted object will drawn in the exactly same location as copied one and will be selected automatically so you can move it to a new location.

To **apply color or style** to the object select it and use Format menu or Format tool bar buttons to change color, thick, dotted and snap to price styles. Note that you can also select color and style of the object before drawing new object: simply deselect previous object (if any), change color / style selections and draw new object.

To **modify properties** of the object – either double click it or use **Edit->Properties** menu or **Alt+ENTER** key

To **delete all** objects use **Edit->Delete All** menu

### Further information

To learn more about drawing tools please read [Drawing tools reference](#) chapter.

## How to use drag-and-drop charting interface

### Introduction

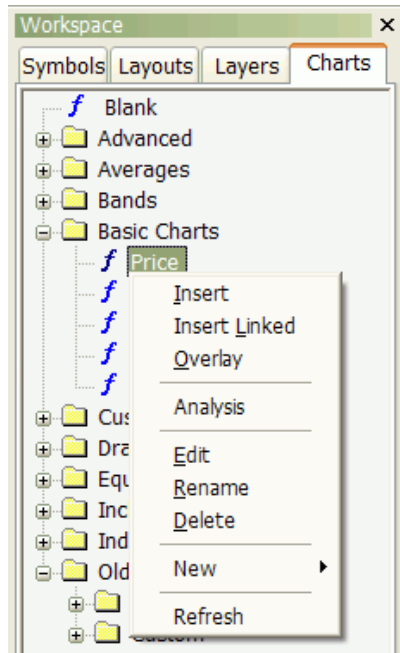
AmiBroker allows you to easily create and modify your indicators with few moves of a mouse. From now on you can build sophisticated indicators without any programming knowledge at all. The available (ready-to-use) indicators are listed in **Charts** tab of the **Workspace** window.

There is a video tutorial at: <http://www.amibroker.net/video/dragdrop1.html> that shows basic usage of new

drag and drop functionality.

#### **How to insert a new indicator.**

To display a new indicator in a separate chart pane just find the indicator in the Charts list (use **View -> Charts** menu) and **double-click** on the indicator name.



Alternatively you can choose **Insert** from the [context menu](#). As a result new indicator pane will be created and **Parameters** dialog will be displayed. Here you can change the properties of the indicator (like color or periods). To accept the settings press **OK** button. (you will find the detailed description of parameters window below).

#### **Example:**

To insert RSI pane – find RSI indicator in the list, double-click on the name, select the number of periods and color, then press OK.

#### **How to overlay one indicator on another indicator.**

To overlay one indicator on another one, press **LEFT** mouse button on the indicator name, drag (with mouse button held) the chosen indicator into the destination pane and release the button.

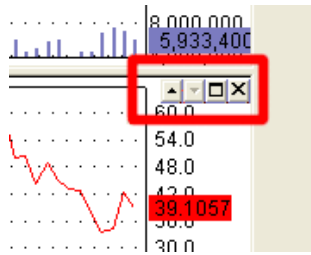
#### **Example:**

To insert another RSI (based different periods number) into the same pane – drag RSI into the previously created RSI pane, change the number of periods in the Parameters window and press OK

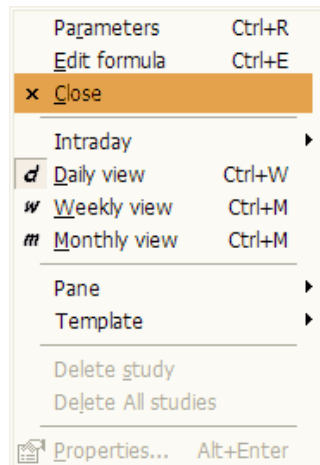
Alternatively you can choose **Overlay** option from [context menu](#).

#### **How to delete the indicator.**

To remove the indicator, press **Close** button from the menu on the top right-hand side of the indicator pane (the menu will be displayed if you place the mouse cursor in the nearby). This menu allows you also to move the indicator pane up/down or maximize the pane.

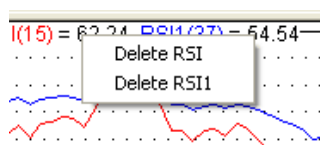


You can also use **Close** command from context menu that shows up when you click on the chart pane with right mouse button.



#### **How to remove the indicator plot from the pane.**

To remove one of the indicators displayed in the indicator pane – click with RIGHT mouse button on the chart title (near the top of chart pane) and select the indicator that you want to remove.



You can also remove the indicator plot using **Delete Indicator** option from [chart context menu](#).

#### **How to change parameters/colors/styles of indicators.**

The **Parameters** window allows you to change parameters, colors and styles of your indicators. Parameters window is displayed when you insert a new indicator. You can also click RIGHT mouse on the chart pane and choose Parameters from the context menu. Parameters window displays all the parameters defined in AFL code of certain indicators (also user-defined parameters) so it's contents depends on the indicator chosen. However – for most of the indicators you will see:

- **Price Field** – the data used to calculate the indicator. If the 'Price Field' contains 'Close', it means that indicator is calculated out of Close prices. Price Field is not available for all indicators, because not all indicators allow you to choose the input (e.g. ADLine).
- **Periods** – defines the number of periods used to calculate the indicator



- **Color** – allows you to change the color of the indicator
- **Style** – allows you to determine the style of the plot (the styles are described in more detail in [Using graph styles and colors](#) tutorial section).

#### How to overlay indicators with different scales.

To have in one pane two (or more) indicators that use different scaling, drag the second indicator onto the first one, in Parameters window click on **Style** field and check **StyleOwnScale** setting.

##### Example:

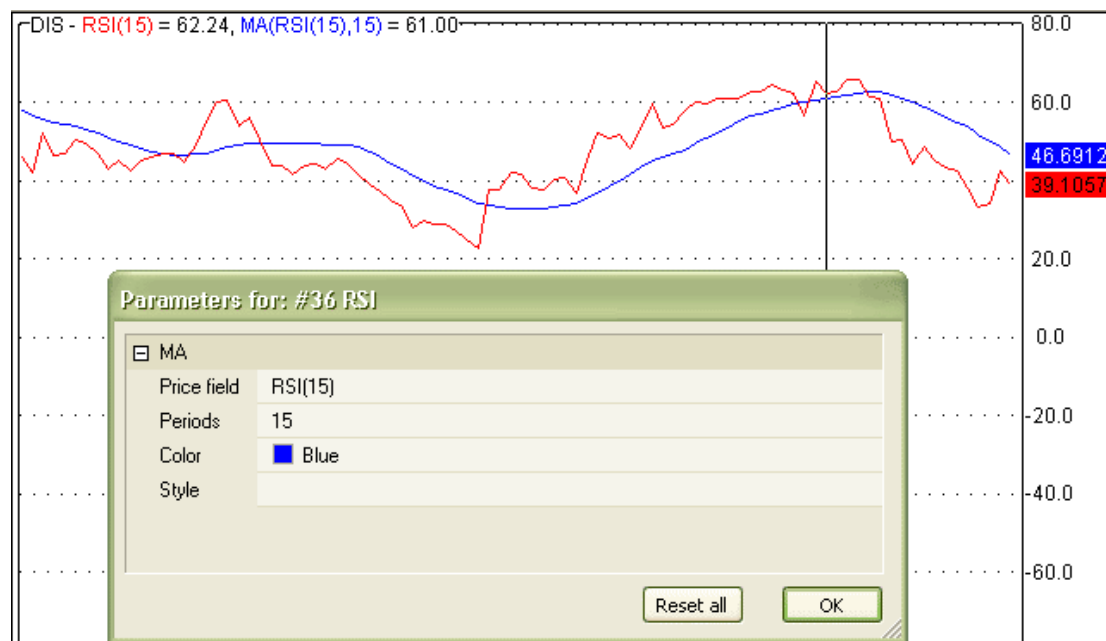
Drag OBV (On Balance Volume) into RSI pane. Then define style as styleOwnScale. As a result – both indicators are visible and properly displayed.

#### How to create an indicator based on another indicator.

AmiBroker allows you also to easily create indicators based on values of another indicator. All you need to do is to press LEFT mouse button on the indicator name, drag (with mouse button held) the chosen indicator into the destination pane and release the button. As a result – the indicator will be placed in the existing chart pane. In the parameters dialog **Price field** parameter indicates what base values are used to calculate the indicator.

##### Example:

To calculate Simple Moving Average of previously created RSI indicator, drag the MA indicator into RSI pane. The contents of "Price Field" parameter indicates, that Moving Average is calculated out of RSI(15) values. (See the below picture).



NOTE: The part below contains technical information for advanced users only. Beginners may skip this part.

#### Using Param(), ParamColor(), ParamToggle(), ParamStyle() functions

These functions, when used in formula, allow you to change indicators' settings directly from **Parameters** window.

#### How to use drag-and-drop charting interface

**Param( "name", defvalue, min = 0, max = 100, step = 1, sincr = 0 )**

Adds a new user-definable parameter, which will be accessible via Parameters dialog.

- "name" – defines parameter name that will be displayed in the parameters dialog
- defvalue – defines default value of the parameter
- min, max – define minimum and maximum values of the parameter
- step – defines minimum increase of the parameter via slider in the Parameters dialog
- sincr – defines the increase of default value when more than one section of the same kind is inserted (dropped) onto the chart. For example if you insert the default Moving Average indicator into the same pane twice, the first moving average will be based on 15 periods, the other one on 25 (defvalue=15 + sincr=10)

**ParamColor( "name", defaultcolor )**

Adds a new user-definable color parameter, accessible via Parameters dialog.

- "name" – defines parameter name that will be displayed in the parameters dialog
- defaultcolor – defines default color value of the parameter

ParamColor function allows you to use **colorCycle** as a default value. When you use colorCycle parameter, default color cycles through red, blue, green, turquoise, gold, violet, bright green, dark yellow, when you insert your indicators into the same pane.

**ParamStyle("name", defaultval = styleLine, mask = maskDefault )** – allows to select the styles applied to the plot from the Parameters window. Apart from styles available in previous versions of AmiBroker, there are two new style constants:

- styleHidden – a combination of styleNoDraw | styleNoRescale
- styleDashed – dashed line

The list of available styles displayed in the Parameters window depends on the **mask** parameter.

- maskDefault – show thick, dashed, hidden, own scale styles (this is default mask for ParamStyle)
- maskAll – show all style flags
- maskPrice – show thick, hidden, own scale, candle, bar
- maskHistogram – show histogram, thick, hidden, own scale, area

**ParamField("name", field = 3 )** – allows to pick the **Price field** for the indicator (field which is used to calculate values of the indicator). Function returns the array defined by *field* parameter. Default value = 3 returns Close array. The possible values of *field* parameter are:

- -1 – ParamField returns the values of the indicator that was inserted as a first one into the pane, or Close if no indicator was present
- 0 – returns **Open** array
- 1 – returns **High** array
- 2 – returns **Low** array
- 3 – returns **Close** array (default)
- 4 – returns **Average** array = (H+L+C)/3
- 5 – returns **Volume** array
- 6 – returns **Open Interest** array
- 7,8,9,... – return values of indicators inserted into the pane.

**ParamToggle("name", "values", defaultval=0 )** – function that allows to use boolean (Yes/No) parameters.

- "name" – the name of the parameter
- "values" – parameter values (separated with | character, e.g. "No|Yes" – first string represents false value and second string represents true value)
- defaultval – default value of the parameter

—

**Example:**

The below indicator allows you to check how the parameters work in the custom code. You can change settings from Parameters dialog.

```
Buy = Cross(MACD(), Signal() );
Sell = Cross(Signal(), MACD() );

pricefield = ParamField("Price Field", 2);
Color = ParamColor("color",colorRed);
style = ParamStyle("style",styleLine,maskAll);
arrows = ParamToggle("Display arrows", "No|Yes",0);
Plot(pricefield,"My Indicator",Color,style);
if(arrows)
{
    PlotShapes(Buy*shapeUpArrow+Sell*shapeDownArrow,IIf(Buy,colorGreen,colorRed) );
}
```

**Special functions: `_SECTION_BEGIN`, `_SECTION_END`, `_SECTION_NAME`, `_DEFAULT_NAME`, `_PARAM_VALUES` explained (for advanced users only)**

These are new functions that are used by drag & drop mechanism. The most important pair is `_SECTION_BEGIN("name")` and `_SECTION_END()`.

When you drop the formula onto chart pane AmiBroker appends the formula you have dragged at the end of existing chart formula and wraps inserted code with `_SECTION_BEGIN("name")` and `_SECTION_END()` markers:

So, if original formula looks as follows:

```
P = ParamField("Price field",-1);
Periods = Param("Periods", 15, 2, 200, 1, 10 );
Plot( MA( P, Periods ), _DEFAULT_NAME(), ParamColor( "Color", colorCycle ),
ParamStyle( "Style" ) );
```

it will be transformed by AmiBroker to:

```
_SECTION_BEGIN( "MA" );
P = ParamField("Price field",-1);
Periods = Param("Periods", 15, 2, 200, 1, 10 );
Plot( MA( P, Periods ), _DEFAULT_NAME(), ParamColor( "Color", colorCycle ),
ParamStyle( "Style" ) );
_SECTION_END();
```

`_SECTION_BEGIN/_SECTION_END` markers allow AmiBroker to identify code parts and modify them later (for example remove individual sections). In addition to that sections provide the way to make sure that parameters having the same name in many code parts do not interfere each other. For example if you drop two moving averages the resulting code will look as follows:

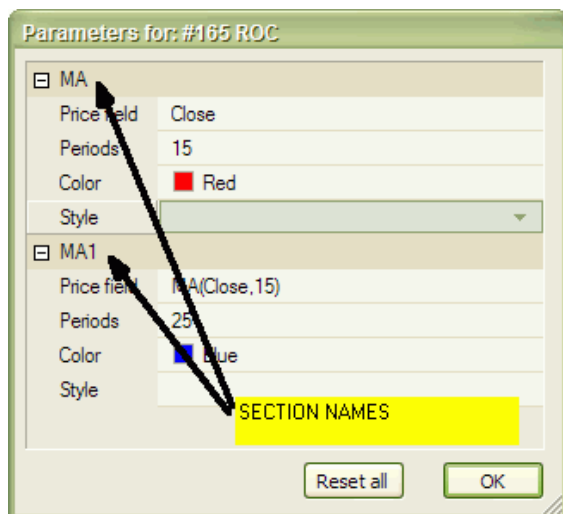
```

_SECTION_BEGIN( "MA" );
P = ParamField( "Price field",-1);
Periods = Param( "Periods", 15, 2, 200, 1, 10 );
Plot( MA( P, Periods ), _DEFAULT_NAME(), ParamColor( "Color", colorCycle ),
ParamStyle( "Style" ) );
_SECTION_END();

_SECTION_BEGIN( "MA1" );
P = ParamField( "Price field",-1);
Periods = Param( "Periods", 15, 2, 200, 1, 10 );
Plot( MA( P, Periods ), _DEFAULT_NAME(), ParamColor( "Color", colorCycle ),
ParamStyle( "Style" ) );
_SECTION_END();

```

Note that code and its parameter names are identical in both parts. Without sections the parameters with the same name will interfere. But thanks to uniquely named sections there is no conflict. This is so because AmiBroker identifies the parameter using section name AND parameter name, so if section names are unique then parameters can be uniquely identified. When dropping indicator AmiBroker automatically checks for already existing section names and auto-numbers similarly named sections to avoid conflicts. Section name also appears in the Parameter dialog:



Last but not least: you should NOT remove `_SECTION_BEGIN` / `_SECTION_END` markers from the formula. If you do, AmiBroker will not be able to recognize sections inside given formula any more and parameters with the same name will interfere with each other.

`_SECTION_NAME` is a function that just gives the name of the function (given in previous `_SECTION_BEGIN` call).

`_DEFAULT_NAME` is a function that returns the default name of plot. The default name consists of section name and comma separated list of values of numeric parameters defined in given section. For example in this code:

```

_SECTION_BEGIN( "MA1" );
P = ParamField( "Price field" );
Periods = Param( "Periods", 15, 2, 200, 1, 10 );
Plot( MA( P, Periods ), _DEFAULT_NAME(), ParamColor( "Color", colorCycle ),

```

```
ParamStyle("Style") );
_SECTION_END( );
```

\_DEFAULT\_NAME will evaluate to "MA1(Close,15)" string.

\_PARAM\_VALUES works the same as \_DEFAULT\_NAME except that no section name is included (so only the list of parameter values is returned). So in above example \_PARAM\_VALUES will evaluate to "(Close, 15)" string.

### **Frequently Asked Questions about drag & drop functionality**

#### **Q. What is the difference between Insert and Insert Linked option in chart menu?**

A. **Insert** command internally creates a copy of the original formula file and places such copy into hidden drag-drop folder so original formula will not be affected by subsequent editing or overlaying other indicators onto it. Double clicking on formula name in the chart tree is equivalent with choosing **Insert** command from the menu. On the other hand **Insert Linked** command does not create any copy of the formula. Instead it creates new chart pane that directly links to original formula. This way subsequent editing and/or overlaying other indicators will modify the original

#### **Q. I can not see buy/sell arrows from my trading system**

A. Trade arrows can be displayed on any chart pane (not only one built-in price chart). However, by default, the arrow display is turned OFF. To turn it ON you have to open Parameter dialog, switch to "Axes and grid" and switch "Show trading arrows" option to "Yes".



#### **Q. The read me says: "Automatic Analysis formula window is now drag&drop target too (you can drag formulas and AFL files onto it)". What does it mean?**

A. It means that you can drag the formula from either Chart tree or .AFL file from Windows Explorer and drop it onto Automatic Analysis (AA) formula window and it will load the formula into AA window. This is an alternative to loading formula via "Load" button in AA window.

#### **Q. Can I drop a shortcut onto the formula window ?**

A: No you can't. You can only drag & drop files with .AFL extension (shortcuts in Windows have .lnk extension).

**Q. Can I add my own formulas to the Chart tree ?**

A. Yes you can. Simply save your .AFL formula into Formulas subfolder of AmiBroker directory and it will appear under "Charts" tree (View->Refresh All may be needed to re-read the directory if you are using external editor)

**Q. I have added new file to the Formulas folder, but it does not show up in the Charts tree unless I restart AmiBroker? Is there a way to refresh Chart tree ?**

A. You can refresh Chart tree by choosing **View->Refresh All** menu.

**Q. If I modify the formula that ships with AmiBroker will it be overwritten by next upgrade?**

A. Yes it will be overwritten. If you wish to make any modifications to the formulas provided with AmiBroker please save your modified versions under new name or (better) in your own custom subfolder.

**Q. I can see Reset All button in Parameters dialog but it sets all parameters to default values. Is there a way to reset SINGLE parameter ?**

A. No, there is no such option yet, but it will be added in upcoming betas.

**Q. I dragged RSI to the price chart pane and got a straight red line at the bottom of the pane. What is wrong?**

A. When you drop two indicators / plots that have drastically different values you have to use style OwnScale for one of it. You can turn on OwnScale style using Parameter dialog. This ensures that scales used for each are independent and you can see them properly. Otherwise they use one common scale that fits both value ranges that results in flattened plots.

**Q. The light grey color of the new AFL special functions\_SECTION\_BEGIN etc makes them invisible in my bluegrey background IB color. How could I change the special functions color ?**

A. Right now, you can't. But there will be a setting for coloring special functions in the next version.

**Q. When I drop the indicator the Parameter dialog does not show all parameters. Is this correct ?**

A. Yes it works that way. The idea behind it is simple. When you drop new indicator AmiBroker displays a dialog with parameters ONLY for currently dropped indicator. This is to make sure that newly inserted indicator parameters are clearly visible (on top) and new user is not overwhelmed by tens of other parameters referring to previously dropped indicators. On the other hand when you choose "Parameters" item from context menu then ALL parameters will show up – allowing you to modify them all any time later.

## User interface customization

A newly introduced customizable user-interface has several nice features that allow complete control over look and feel of AmiBroker user interface.

### Advanced nested docking / tear-off tabs



To dock a pane into any side of the application or as a tab simply click on docking window caption bar and drag it. If you do this, docking stickers will show up to make it easy to choose destination place as shown below



You can also click on docking pane tab and drag it (tear off) and dock as a separate window. This way you can arrange all docking windows either as separate windows or as tabs or as a mixture of these two approaches. You can also make window / tab floating if you drag it while holding down CTRL key.

### Sliding Auto-hide panes

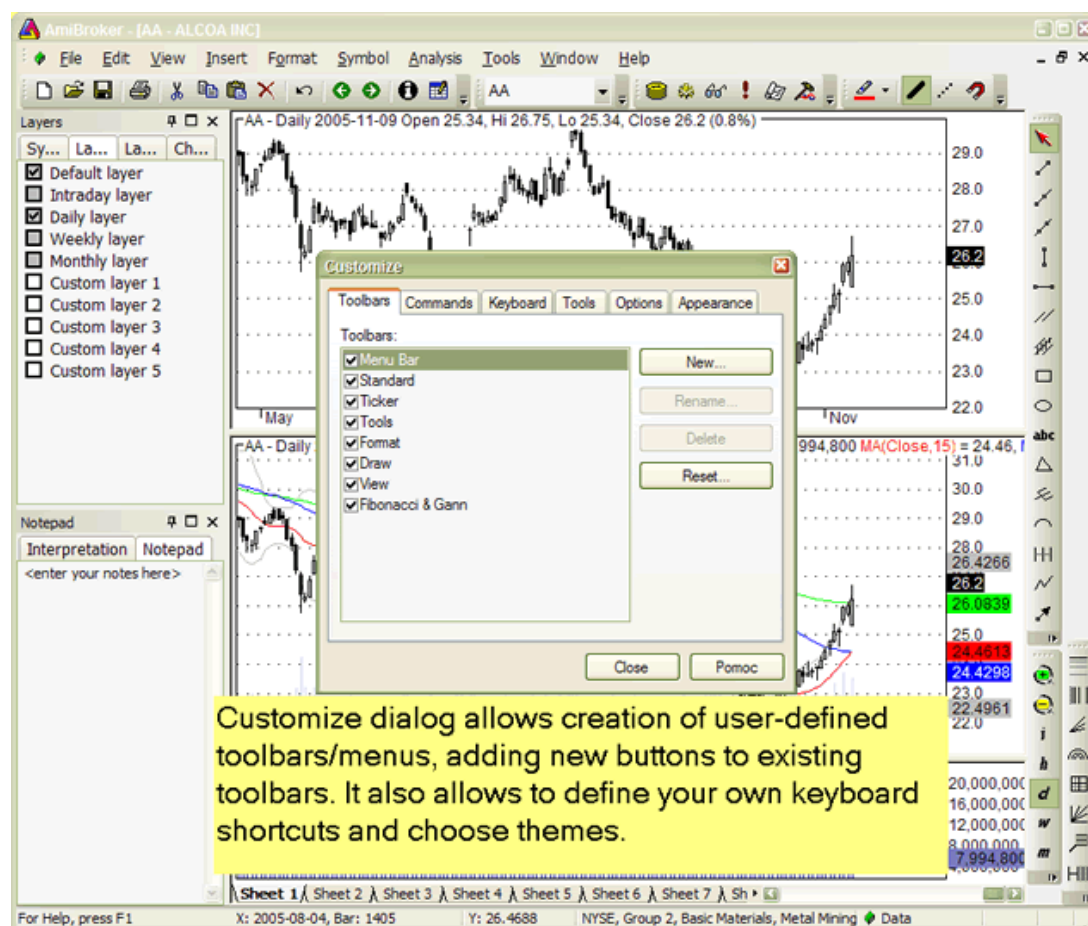
Another very useful feature that allows to conserve precious real estate on your monitor is auto-hiding of panes. To control (switch on/off) this feature there is a pinup button in the upper right corner of each docking window. If you unpin it – the pane will automatically hide when it loses focus.





## Advanced customizable toolbars, menus and keyboard shortcuts

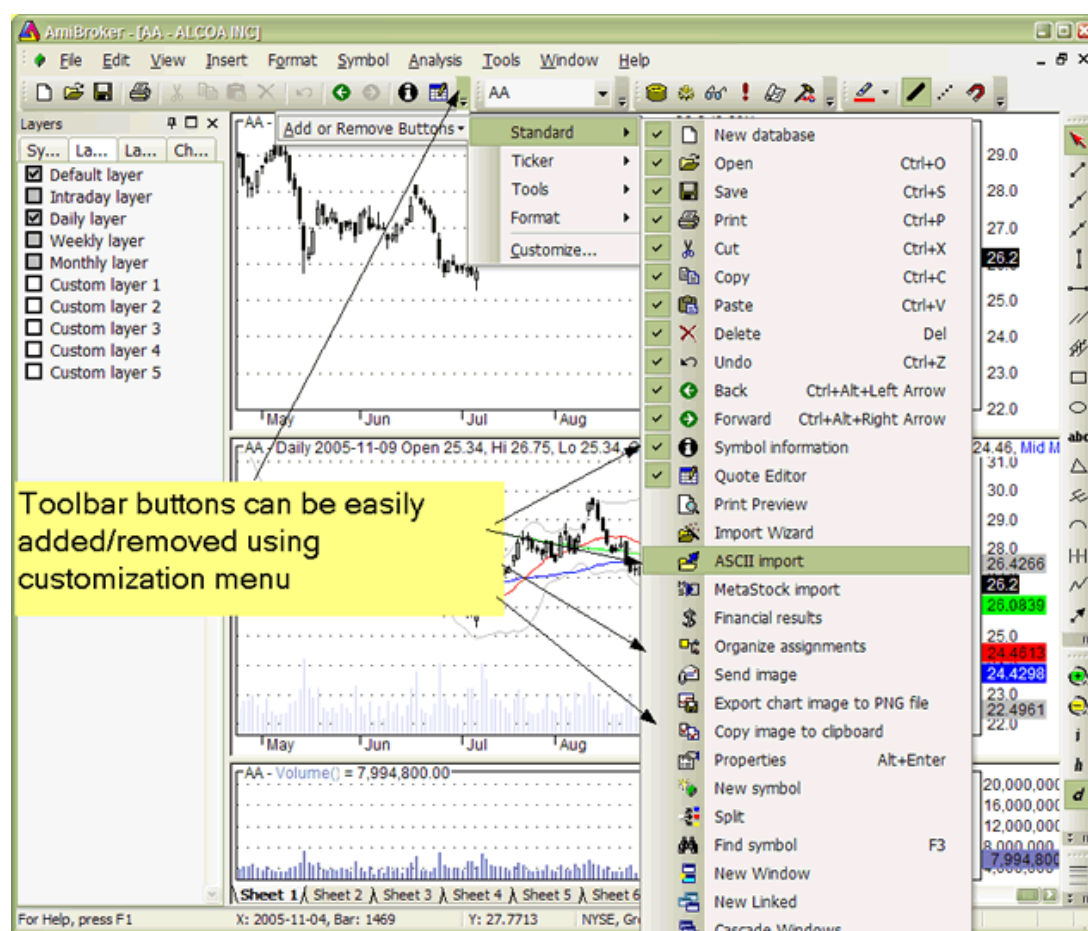
New user interface allows full user control over appearance, layout and position of all toolbars, buttons and menus. It allows you to add your own buttons, remove/re-arrange existing ones. Also you define or re-define new/existing keyboard shortcuts. All these customization features are available from **Tools->Customize** menu or from **Customize** chevron menu.



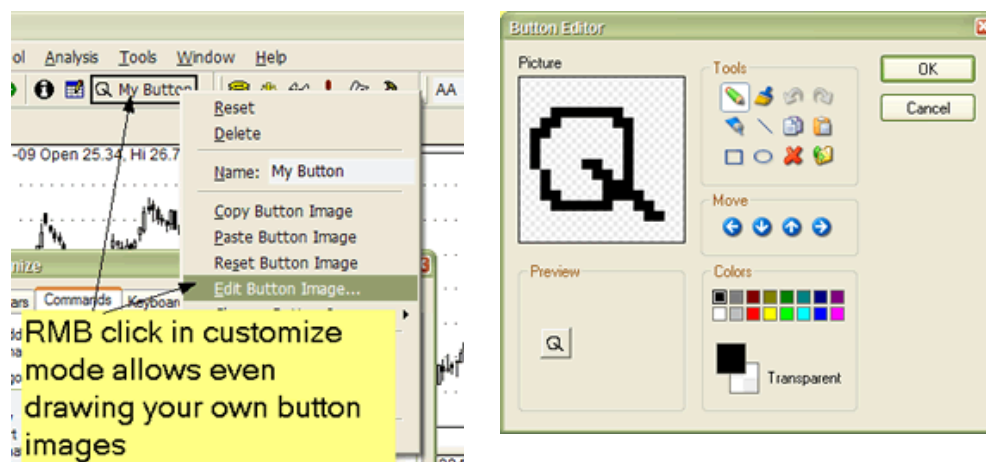
Chevron menu is available from little arrow button placed at the end of toolbar strip. It allows to access auto-hidden elements of the toolbar as well as customization features.



Add or Remove buttons submenu allows to quickly show / hide toolbar buttons according to your preference. In customization mode (when you enter it using Tools→Customize you can also move buttons around to change the order in which they appear, and you can also resize edit fields and combo fields (such as ticker selection field) by selecting them first and resizing the border that will show after making selection.



You can even add and design your own buttons using built-in image editor:



## Themed appearance

AmiBroker allows also to pick your preferred user-interface "appearance" or "theme" to suit your personal taste.



### MDI (multiple document interface) tabs



AmiBroker is multiple document interface (MDI) application. In short it means that it allows you to open and work with multiple windows at the same time. To learn more about what MDI is you may check this article: [http://en.wikipedia.org/wiki/Multiple\\_document\\_interface](http://en.wikipedia.org/wiki/Multiple_document_interface)

Now MDI tabs (shown in the picture above) are just an additional way to switch multiple open windows (in addition to **Window** menu where the list of open document windows is also available).

It is important to understand that MDI tabs are **not** "user definable" in the sense that you can not define their names freely, unlike [chart sheets \(which are definable\)](#). Their names are automatically derived from

document/window name. For chart windows the name is always in the format of: Symbol – FullName, web browser windows use HTML page title (as defined by HTML document), account manager windows use actual account file name (that you can choose when you save them).

MDI tabs are basically document window switcher (like Windows TASK BAR in the bottom) and they are automatically managed by AmiBroker whenever you open new or close window.

And it works exactly using the same idea as Windows task bar. Let us look at this analogy closer:

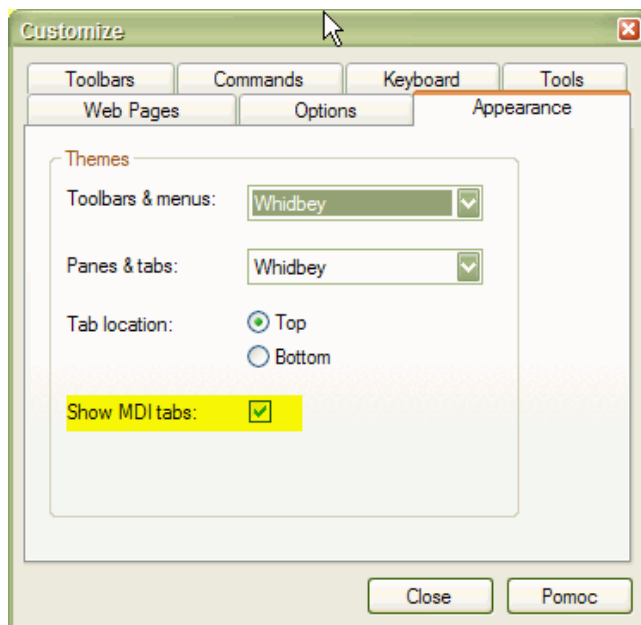
When you use Windows Task Bar:

- you open the **application** – a new button in the task bar appears
- and you can switch between open **applications** using task bar buttons.
- you can not rename the button because it represents **application** name.
- and you need to be careful with opening too many applications because all open applications consume system resources

Now using AmiBroker MDI tabs:

- you open the document (**window**) → a new button (tab) appears
- you can switch between open **windows** using buttons (tabs)
- you can not rename the button because it represents document/**window** name
- and you need to be careful with opening too many documents/**windows** because all open documents consume system resources

You can **turn off** MDI tabs by unchecking "Show MDI tabs" box in the Tools→Customize, Appearance page, as shown below:



Historical note: In pre-4.90 versions, to switch the documents you would need to use Window menu. Now in addition to that you can use tabs. But this is just convenience feature, more info at:

[http://en.wikipedia.org/wiki/Tabbed\\_Document\\_Interface](http://en.wikipedia.org/wiki/Tabbed_Document_Interface) (Note that wikipedia links describing TDI / MDI are somewhat outdated and AmiBroker actually combines advantages of BOTH TDI and MDI approaches (for example you can tile windows in AB's TDI))



For more information see Houston conference presentation: <http://www.amibroker.com/docs/Houston1.pdf> (PDF format), <http://www.amibroker.com/docs/Houston1.html> (Flash format).

## Working with chart sheets and window layouts

AmiBroker manages multiple chart sheets and **multi-window layouts** with ability to quickly load/save them. This feature enables you to quickly switch between different indicator sets saving your time dramatically.

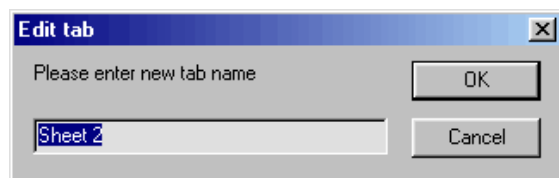
### Chart sheets and templates

A chart sheet is a set of chart panes (with indicators) displayed within single frame.

You can switch between different sheets by clicking on the tabs located in the bottom of AmiBroker window as show in the following picture:

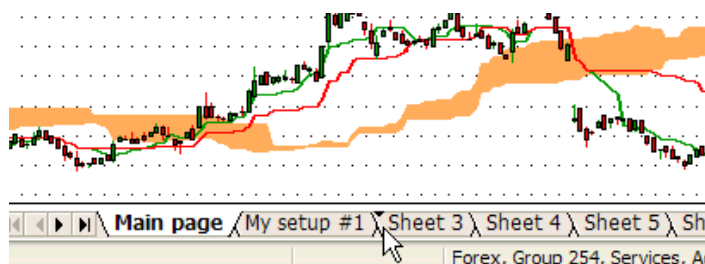


You can change the name of the tab by clicking on it with RIGHT mouse button, so the following window appears:

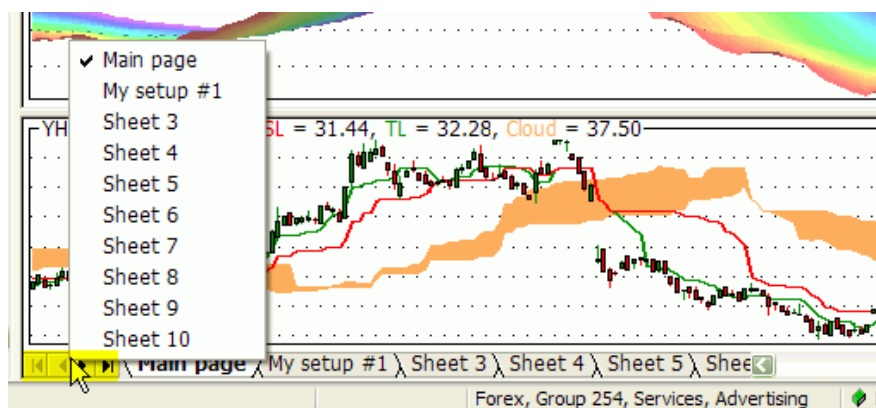


You can change all four tab names (one by one) so they are more descriptive (and they relate to the contents of the sheet).

You can scroll tabs using arrow buttons and you can re-arrange them by dragging (click on tab, hold down left mouse button and drag to desired position – an arrow will show target position).



You can also access any sheet quickly by clicking with RIGHT mouse button over arrows to pop-up the menu that lists all tabs and allows immediate selection (without scrolling)



The next step is to set up your sheets according to your personal preference. Just add/remove chart panes to/from each sheet. This way you can have up to 60 different indicator sets that you can recall very quickly by switching to appropriate tab. The actual number of sheets is definable in **Tools->Preferences->Charting "Number of chart sheets"**

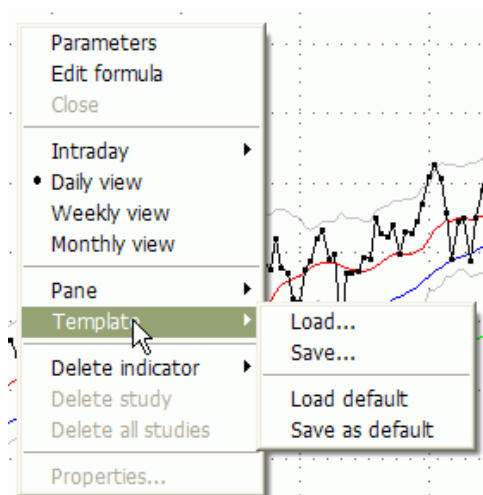
The complete set of chart sheets is called a "template" and you can make this setup permanent just right-click on the chart and select the following menu item (**Template->Save, Template->Save as default**):

The default template is used if you create a new window (**Window->New**)

You can also load once saved template by choosing **Template->Load** from chart's right mouse button menu.

In addition to old local template format a new one is added with .chart extension that keeps not only window sizes and formula references (paths) but also formulas themselves, so all you need to do is to save your chart into one file (Chart Template, Complete \*.chart) and copy that file onto different computer and chart will be recreated with all formulas linked to it.

*To Save chart into new format do the following:*



1. Click with RIGHT MOUSE button over the chart and select **Template->Save...**



2. In the file dialog, "**Files of type**" combo select "**Chart Template, Complete (\*.chart)**"
3. Type the file name and click **Save**.

*To load previously saved complete chart do the following:*

1. Click with RIGHT MOUSE button over the chart and select **Template-->Load...**
2. In the file dialog, select previously saved \*.chart file and press "**Open**"

Note: The procedure AmiBroker does internally is as follows: When you save the chart into new format it saves XML file with:

- a) names of all sheets, panes, their sizes, locations and other settings
- b) paths to all formulas used by all panes
- c) the text of formulas themselves

When you load the chart in new format AmiBroker:

- a) sets up the sheets/panes according to information stored in the file
- b) for each formula stored in the file it checks if the same formula exists already on target computer:
  - if it does not exist – it will create one
  - if it exists and the contents is identical to the formula stored in .chart file it will do nothing
  - if it exists and the contents is different then it will create NEW formula file with \_imported.afl suffix (so old file is not touched) and will reference the pane to the \_imported.afl formula instead.

**IMPORTANT NOTE:** if you use any #include files AmiBroker will store the contents of include files as well inside chart file and will attempt to recreate them on target machine. Please note that in case of includes it will check if it exists and if it is different. If both conditions are met (different file exists already) it will ask to replace or not. If you choose to replace – it will replace and make backup of existing one with .bak extension. If you are using any files in "standard include files" and include them using <> braces, AmiBroker will restore files in target machine standard include folder as well (even if the standard include folder path is different on the source machine).

A new .chart format is intended to be used to port charts between different computers. For storing layouts/templates on local computer you should rather use old formats as they consume much less space (they store only references, not the formulas themselves). One may however use new format for archiving purposes as it keeps formulas and all references in one file that is very convenient for backups.

## Window layouts

A window layout is a complete set of multiple windows open each with different symbol, different display interval, different size, different set of chart sheets.

The picture below shows 4-window layout each with different set of indicator panes. To the left you can see "Layouts" pane in the Workspace window showing the list of stored local and global layouts.



Using AmiBroker 4.20 you can now have unlimited number of custom, multiple-window templates that can be switched between with just double click on layout name in the "**Layouts**" tab of the Workspace window.

You can **open**, **save**, **delete** layout by clicking on the **Layout** tree with right mouse button and choosing appropriate function. "**Save As**" option saves current layout under new name.

**Local layouts** are per-database while **Global layouts** are visible from all databases.

Information saved in layouts include: window sizes and positions, maximized/minimized state chart panes available on each sheet (independent for each window), selected bar interval, selected symbol, selected chart sheet

Most recently used layout can be saved on exit and database switch automatically (see: **Tools->Preferences->Miscellaneous** "Save on exit: Layouts")

Note: since version 4.90 multiple windows can be switched not only using old-style Window menu but also using new MDI tabs. More on MDI tabs can be found in the "[User-interface customization](#)" chapter.

## Using layers

### What layers are

Layers are like pieces of transparent plastic. You can put drawings on them. Layers can be made visible or invisible. This allows to show/hide drawings placed on given layer without affecting the drawings placed on other layers.

**How to work with layers.**

First of all make sure that Workspace window is visible (View->Workspace)

Then switch to "Layers" tab. Here you can see the list of pre-defined layers.

The checkboxes on the left side of each layer control layer visibility. If checkbox is marked then given layer is visible, if it is unmarked – the layer is invisible. Initially first five layers will be "locked" to intervals.

These built-in layers are:

Default layer – always visible

Intraday layer – visible only when viewing intraday charts

Daily layer – visible only when viewing daily charts

Weekly layer – visible only when viewing weekly charts

Monthly layer – visible only when viewing monthly charts

A locked layer changes its visibility automatically when interval changes and you can not change its visibility by clicking on the left-side checkbox.

The remaining layers are not locked and they can be shown/hidden freely by marking the checkbox.

To draw a study in a given layer simply

a) SELECT the layer first (click on name to highlight it)

b) DRAW the study as usual

As long you select the other layer all drawings will be placed on selected layer. After drawing a study you can assign it to any other layer via object properties box.

**Context menu**

If you click on layer name with right mouse button you will see the context menu containing the following options:

Add layer

Remove layer

Show all layers

Hide all layers

Toggle

Unlock built-in layers

Lock built-in layers

Properties.

Add/Remove layer are self-explanatory. Please note that you can not remove first 5 (built-in) layers

Show all/Hide all – shows and hides all NOT LOCKED layers

Toggle – toggles visibility of all NOT LOCKED layers

Unlock/Lock built-in layers – allows you to unlock/lock 5 first (built-in) layers. Once layer is unlocked its visibility does not change automatically when interval changes and you can show/hide it manually.

Properties – this launches properties box that allows you to rename layer and decide if given layer should or should not be locked to interval displayed.

If you mark "Lock visibility to interval" box the layer will show/hide automatically depending on what interval is currently displayed. You can define visibility for **each** layer using "Interval" combo and "Show/hide automatically" buttons. Note that there is a \*separate\* visibility setting for EACH interval. The layer properties box ALWAYS shows "monthly" interval at start but this is just a startup condition you just switch to particular interval and modify visibility. To setup locked layer completely you have to set visibility for **every layer listed** in the "Interval" combo-box. Simply select the interval and choose if layer should be shown or hidden for this interval, select next interval and again choose show or hide, select next and so on...until you define visibility for all intervals.

**Using layers**



## Using Web Research window

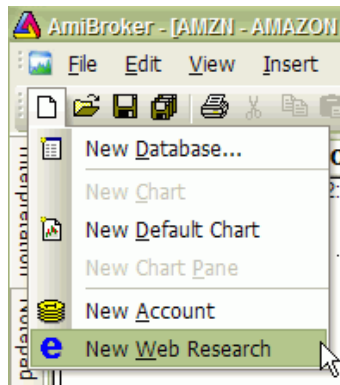
Web Research window allows you to view on-line news, research, profiles, statistics and all kind of information related to currently selected symbol available over the Internet (World Wide Web). Using Web Research instead of plain web browser has speed advantage as you don't need to type complicated/long addresses (URLs) each time you need to get desired information.

Web Research window introduced in version 4.90, replaces and enhances previously available [Profile window](#). Now it allows unlimited number of user-definable web research (profile) pages, browsing to any web page (just type URL), tab-browsing, opening multiple pages at once, selective auto-synchronization.

Web-Reasarch uses Internet Explorer engine so you can be sure that pages are rendered with the same quality you would get from stand-alone browser.

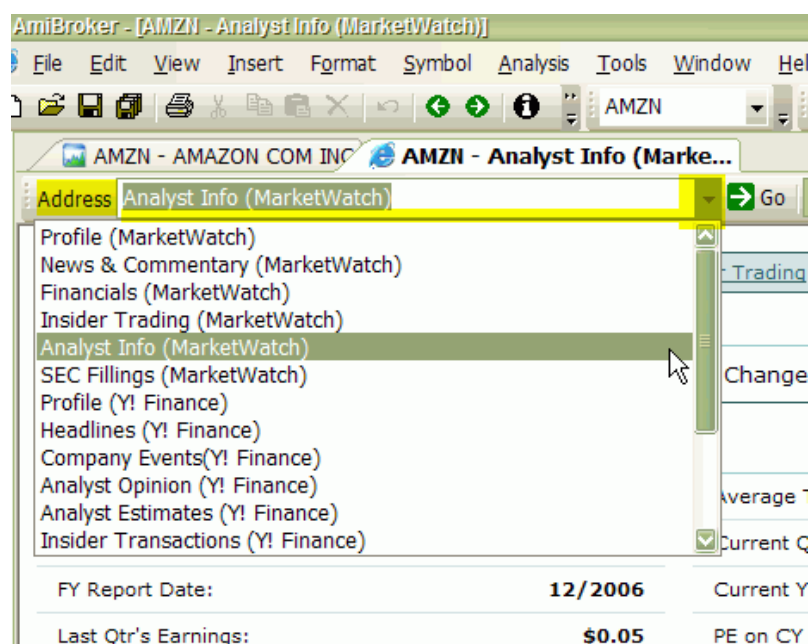
### OPEN NEW WEB RESEARCH WINDOW

Use **File->New->Web Research** menu to create new web research window

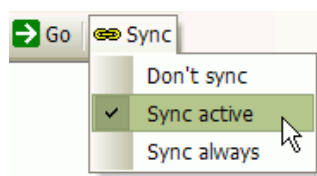


### PICKING PRE-DEFINED WEB RESEARCH PAGE:

To display any pre-defined web research page, simply click on the drop down arrow in the Address combo-box and pick one item from the list. Once you do so, the web page relevant to currently selected symbol will be automatically displayed.



Now you can specify if and when displayed page should change automatically if you select different symbol.

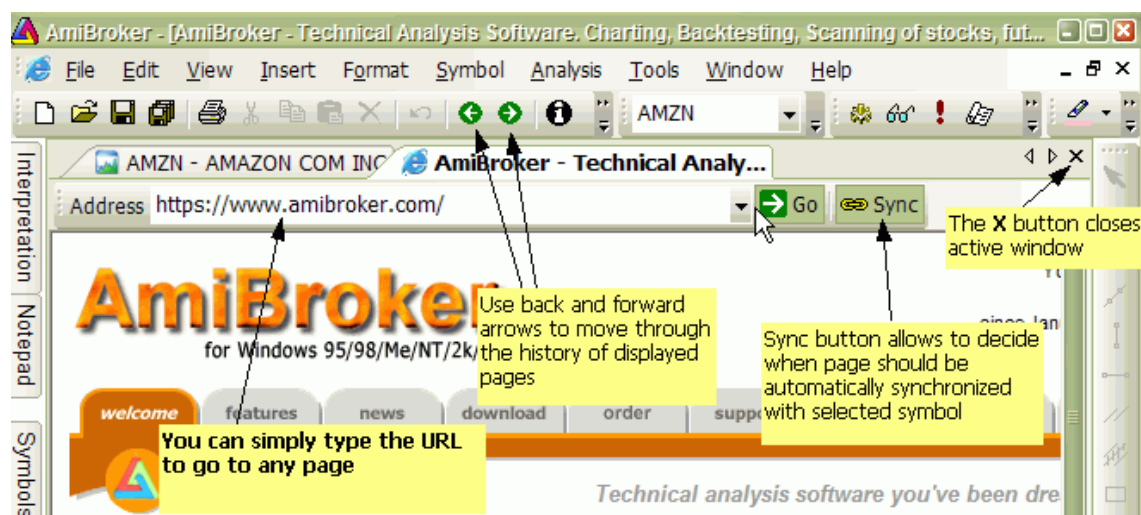


The **Sync** button allows to decide when page should be automatically synchronized with currently selected symbol.

- **Don't sync** – means that page should not be synchronized with currently selected symbol at all
- **Sync active** – means that page should be synchronized ONLY when it is currently active or becomes active (by user clicking on given tab) – this is recommended setting for web-research profiles since it conserves bandwidth and resources (not active pages are not synchronized and do not consume any bandwidth)
- **Sync always** – means that page is synchronized with currently selected symbol always, no matter if it is active or not.

## NAVIGATION

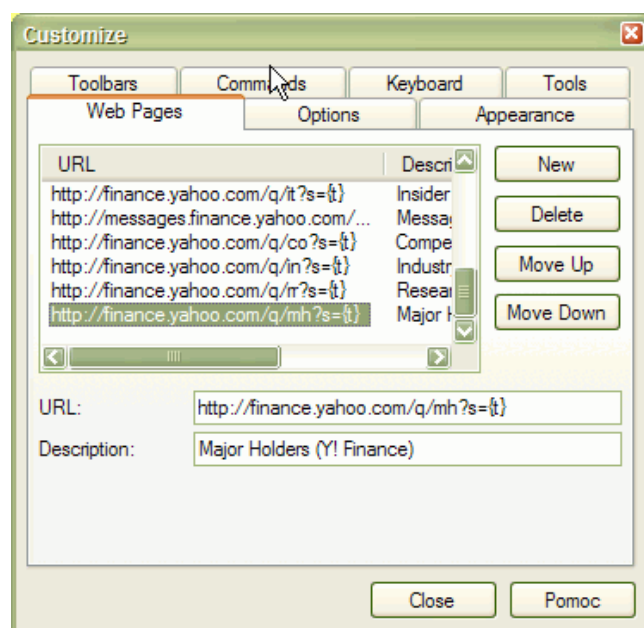
Web Research window operates in a way very similar to stand-alone browser. To display any web page just type the URL address to "Address" field and press ENTER (RETURN) key. To navigate back and forward in the history use <- and -> buttons.



To close currently displayed page use regular window close **X** button as shown in the picture above

## DEFINING YOUR OWN WEB RESEARCH PLACES

In addition to web-research pre-defined pages you can define any number of your own places. To do so use **Tools->Customize** menu, **Web Pages** tab.



To add new place press **New** button, then type the URL template in the **URL** field and web page description in the **Description** field.

The URL template is the web address in that has parts that depend on selected symbol. The URL template is parsed by AmiBroker to make actual URL to the web page. For example to see Yahoo's profiles page you can use following URL template:

`http://biz.yahoo.com/p/{t0}/{t}.html.`

Symbols enclosed in brackets {} define fields which are evaluated in execution time. {t0} symbol is evaluated to the first character of the ticker name and {t} is evaluated to the whole ticker name. So if AAPL is selected AmiBroker will generate following URL from above template:

`http://biz.yahoo.com/p/a/aapl.html`

Then AmiBroker uses built-in web browser (Web Research window) to display the contents of the page.

### Special fields encoding scheme

As shown in above example template URL can contain special fields which are substituted at run time by values corresponding to the currently selected symbol. The format of the special field is {x} where x is describes field type. Currently there are three allowable field types: ticker symbol in original case {t}, ticker symbol in lowercase {s}, ticker symbol in UPPERCASE {S}, alias {a}, web id {i}. You can specify those fields anywhere within the URL and AmiBroker will replace them with appropriate values entered in the Information window. You can also reference to single characters of ticker, alias or web id. This is useful when given web site uses first characters of, for example, ticker to group the html files (Yahoo Finance site does that), so you have files for tickers beginning with 'a' stored in subdirectory 'a'. To reference to single character of the field use second format style {xn} where x is field type described above and n is zero-based index of the character. So {a0} will evaluate to the first character of the alias string. To get first two characters of a ticker write simply {t0}{t1}. Note about web id field: this new field in Information window was added to handle situations when web sites do not use ticker names for storing profile files. I found some sites that use their own numbering system so they assign unique number to each symbol. AmiBroker allows you to use this nonstandard coding for viewing profiles. All you have to do is to enter correct IDs in Web ID field and use appropriate template URL with {i} keyword.

### Pages stored locally

You may want to have all pages stored on your local hard disk. This has an advantage that profiles are accessible instantly but they can take significant amount of storage space and you will need to update them from time to time. To access locally stored files use the following template URL (example, C: denotes drive): `file://C:\the_folder_with_profile_files\{t}.html`. You are not limited to HTML files, you can use simple TXT files instead. Then create (or download) the .html (or txt) files for each symbol in the portfolio. These files should obey the following naming convention: <ticker>.html. So for example for APPLE (ticker AAPL) the profile should have the name AAPL.html (or AAPL.txt)

### Web-based profiles

If you want to display the profiles from remote web pages you will need to find out how they are accessible (the URL to the web page) and how the data for different symbols are accessible. I will describe the problem on the example of Sharenet ([www.sharenet.co.za](http://www.sharenet.co.za)) site providing the data for companies listed on Johannesburg Stock Exchanges. Sharenet provides company information that is accessible at the following address (URL):

`http://www.sharenet.co.za/free/free_company_na.phtml?code=JSECODEp>`

The problem is that database provided by Sharenet uses long ticker names and **JSECODE** is a short symbol code. For example for "Accord Technologies" company the ticker in Sharenet database is ACCORD but the code is ACR. To solve the problem we will need to use **Web ID** field in the symbol Information window. If you have Sharenet database just choose the ACCORD from the ticker list, open *Symbol->Information* window and enter ACR to the **Web ID** edit box and click OK. Then enter the following URL template to the **URL** edit box:



[http://www.sharenet.co.za/free/free\\_company\\_na.phtml?code={i}p](http://www.sharenet.co.za/free/free_company_na.phtml?code={i}p)>

To be 100% sure please select the text above with a mouse. Then copy it to the clipboard (Edit->Copy, CTRL-C). Then switch to AmiBroker and click on the Profile URL edit box. Delete everything from it and press CTRL-V (this will paste the text). Type "Sharenet" into **Description** field.

Please note that we have used **{i}** special field in the template that will be replaced by AmiBroker with the text entered in the Web ID field of the symbol information window. Now please select *File->New->Web Research* and pick Sharenet from Address combo box. You should see the profile for ACCORD company.

You can also delete any entry by selecting it from the list and pressing **Delete** button. You can change the order in which pages appear in the Web Research address combo using **Move Up** and **Move Down** buttons (select the item first and then use buttons).

Configuration data are stored in webpages.cfg plain text file that holds any number of URL templates in the form of:

URLTemplate|Description

(each entry in separate line)

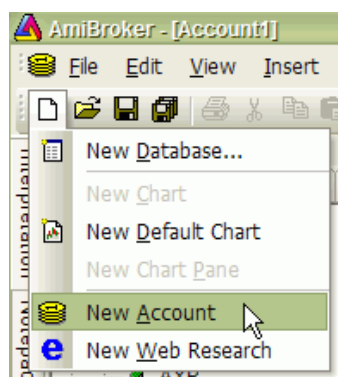
## Using account manager

Account manager is a tool for keeping track of your trades and your performance. You are able to enter trades you make, deposit/withdraw funds, check the statistics and historical performance. All transactions are recorded so you will never forget what happened in the past. Account manager allows you to keep track of unlimited number of accounts.

New account manager replaces and enhances functionality provided by portfolio manager in pre-4.90 versions.

### CREATE A NEW ACCOUNT

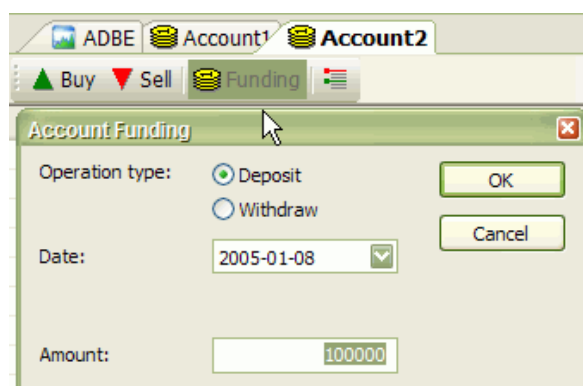
Use File->New->Account menu to create new account



### FUNDING AN ACCOUNT

Before you do any trading, you have to fund your account. To do so press "FUNDING" button on the account manager toolbar, then select "Deposit" as operation type, enter the DATE when you have funded your account and enter the amount.

Note that funding date must PRECEDE any trading, as account manager won't allow you to trade prior to funding date. Initial deposit will show as "initial equity" in summary tab.



### THE SETTINGS

It is good idea to go to "Summary tab" and setup commissions and trading mode. If this account is used for End-of-day trading you should set "EOD Mode" to YES, otherwise (if you trade intraday) you should set "EOD Mode" to NO. Depending on this setting Buy/Sell dialogs will allow you to enter date and time of the

trade or only date.

The screenshot shows the 'Account1' window in AmiBroker. The 'General Settings' section includes fields for Name (My Test Account), End of Day Mode (True), and Default Commission (Per share: 0.0050, % of trade value: 0, Min amount: 1.0000, Min % of value: 0.20%, Max amount: 0, Max % of value: 0). The 'Statistics' section shows Initial Equity, Cash Balance, Net Liquidation Value, Realized Profit/Loss, and Realized % Profit/Loss. The 'All Open Positions' and 'Long Positions Only' sections are also visible. The bottom tab bar shows 'Closed Trades', 'Equity History', and 'Summary' (highlighted). A text box with an arrow points to the 'Summary' tab, stating 'Active sheet can be selected here'.

Commission table allows to enter both per-share (per-contract) commissions and commissions that are expressed as percent of trade value. Or a combination of both. You can also set minimums and maximums expressed in dollar amount and/or percent of trade value. For example if your broker may use 0.01\$ (one cent) per share commission, then you would use PerShare = 0.01 and %OfTradeValue = 0. If your broker uses say 0.2% of trade value then you would use PerShare = 0 and %OfTradeValue = 0.2;

Practical example: Interactive Brokers default commission for U.S. stocks is: 0.005 per share but not less than 1 dollar and not more than 0.2% of trade value. Appropriate settings for such schedule are shown in the screenshot above.

Commission table works as follows: first sum of per-share commission and % of trade value is calculated. Then the result is checked against minimum and maximum limits and if calculated value exceeds the limit then commission is set to value of such the limit, otherwise calculated value is used without change.

Summary page contains a little bit of basic statistics as well.

## ENTERING TRADES

Once you funded an account you can enter trades. To buy (enter long position or cover short position ) click on "BUY" button.

The screenshot shows the 'Buy' dialog box in AmiBroker. The dialog is titled 'Buy' and has a close button (X). It contains the following fields and buttons:

- Ticker: ABE (dropdown menu)
- Date: 2006-01-26 (dropdown menu)
- Price: 38.77 (text box)
- Qty: 500 (text box) with a 'Set to Max.' button
- Net market value: 19385 (text box)
- Commission: 2.5 (text box)
- Margin deposit: (text box) with a note: '(futures only - enter it in Information window)'
- Currency: (text box) with a note: '(foreign /non-base/ currency not implemented yet)'
- FX rate: 1 (text box)
- Buttons: OK, Cancel

The background shows a table with columns ID, Date/Time, Type, and Symbol.

Then in the Buy dialog you need to select the symbol, the trade date/time. Once they are entered AmiBroker will display price of given symbol at the selected date/time (or preceding one if no exact match is found). It will also calculate maximum possible quantity taking price and available funds into account.

You can change the price and quantity manually.

All other values (net market value, commission, market deposit, currency, fx rate) are calculated or retrieved automatically from Symbol->Information page. Once values are good, click OK to confirm transaction. If you made mistake, you can press UNDO (Edit->Undo) to revert last transaction.

Similar procedure is for selling (entering short positions or closing longs) with the exception that you should press "SELL" button instead.

All transactions that you made are listed in the "Transactions" sheet. All open positions are listed in "Open Positions" sheet. If you enter the trade for symbol that has position already open, AMiBroker will adjust "open positions" accordingly (perform scaling in/out). Once open position is closed it is removed from "open positions" list and moved to "Closed trades" sheet.

ID	Date/Time	Type	Symbol	Qty	Price	Net va
2	2006-01-26	Buy	ADBE	500	38.77	193
1	2005-01-08	Deposit	<Funding>	1	100000	1000

Newly entered trade appears in the **Transactions** sheet.

Other sheets show currently **Open Positions**, **Closed Trades**, and **Equity History**

Transactions Open Positions Closed Trades Equ

After each transaction, "Equity history" sheet is updated with current account equity value and also "Summary" page is updated with basic open/long/short trade stats.(More stats are to come).

## IMPORTANT

You have to remember that you must enter all transactions in chronological manner(oldest first, newest last), as account manager won't allow you to add trades out-of-order. If you make mistake, there is one-level undo that you can use to revert to state before last transaction. If you made more mistakes, the only option is to close account without saving and re-open original file.

## SAVING YOUR ACCOUNT DATA

To save edits made to account use File->Save (or File->Save As to save under new name). Note that **account files are NOT encrypted now**, and it is quite easy to read the file for everyone who has the access to it. So make sure not to leave your files on some public computer. Password protection/encryption is planned but NOT implemented yet.

## OPENING PREVIOUSLY CREATED ACCOUNT

To open account file, go to File->Open, in the File dialog, select "Account (\*.acx)" from "Files of type" combo-box, and select the account file you want to load.

## **MULTIPLE ACCOUNTS**

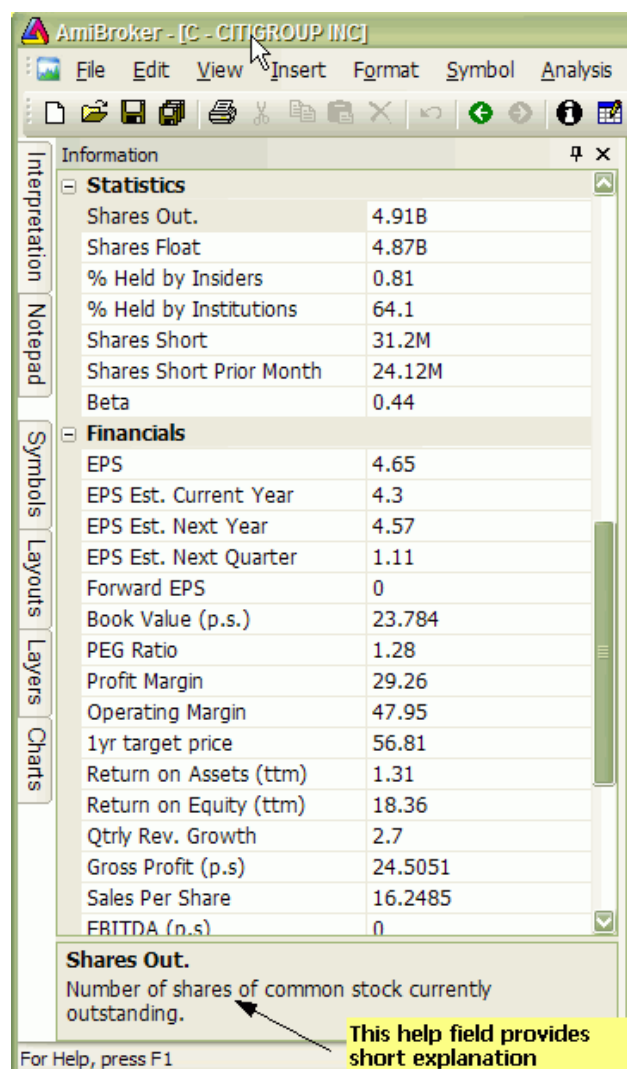
You can create/open multiple accounts at once (just use File->New->Account, File->Open many times).

## Using fundamental data

AmiBroker 4.90 adds ability to use 32 fundamental data items. Fundamental data can be automatically downloaded for all U.S. stocks for free using AmiQuote. New [Information window](#) allows you to view these items, while new AFL function: [GetFnData](#) allows to access fundamentals programmatically.

### INFORMATION WINDOW

To display fundamental data in Information window, please use **Symbol->Information** menu. This will open Information window with several fundamenta data fields as shown in the picture below (if you created new database, it probably will not have these data present initially and you would need to download them)



### DOWNLOADING FREE FUNDAMENTAL DATA FROM YAHOO

New version of AmiQuote now features ability to download free fundamental data from Yahoo Finance web site. This is implemented using 2 different Yahoo pages:

1. **Yahoo Fundamental – Basic** data source (free basic fundamental data, 200 symbols in one request).

Data are retrieved from the following URL: <http://finance.yahoo.com/q?s={Ticker}> (Download data link).

That page provides the following data:

EPS (ttm)  
 EPS Est Current Year  
 EPS Est Next Year  
 EPS Est Next Quarter  
 PEG Ratio  
 Book Value  
 EBITDA  
 Sales Revenue  
 Dividend Pay date  
 Ex Dividend date  
 Dividend Per Share  
 1yr Target Price  
 Shares Float  
 Shares Outstanding

Explanation of values: <http://help.yahoo.com/help/us/fin/quote/quote-03.html>

2. **Yahoo Fundamental – Extra** data source (extended fundamental data, 1 symbol in one request, more data – available in registered version only).

Data are retrieved from the following URL: <http://finance.yahoo.com/q/ks?s={Ticker}> (Key Statistics page)

That page provides following data:

Forward P/E  
 PEG Ratio  
 Profit Margin  
 Operating Margin  
 Return on Assets  
 Return on Equity  
 Revenue (ttm)  
 Qtrly Revenue Growth  
 Gross Profit  
 EBITDA  
 (Diluted) EPS  
 Qtrly Earnings Growth  
 Book Value Per Share  
 Operating Cash Flow  
 Levered Free Cash Flow  
 Beta  
 Shares Outstanding  
 Float  
 % Held by Insiders  
 % Held by Institutions  
 Shares Short (prior month)  
 Shares Short  
 Forward Annual Dividend Rate  
 Trailing Annual Dividend Rate



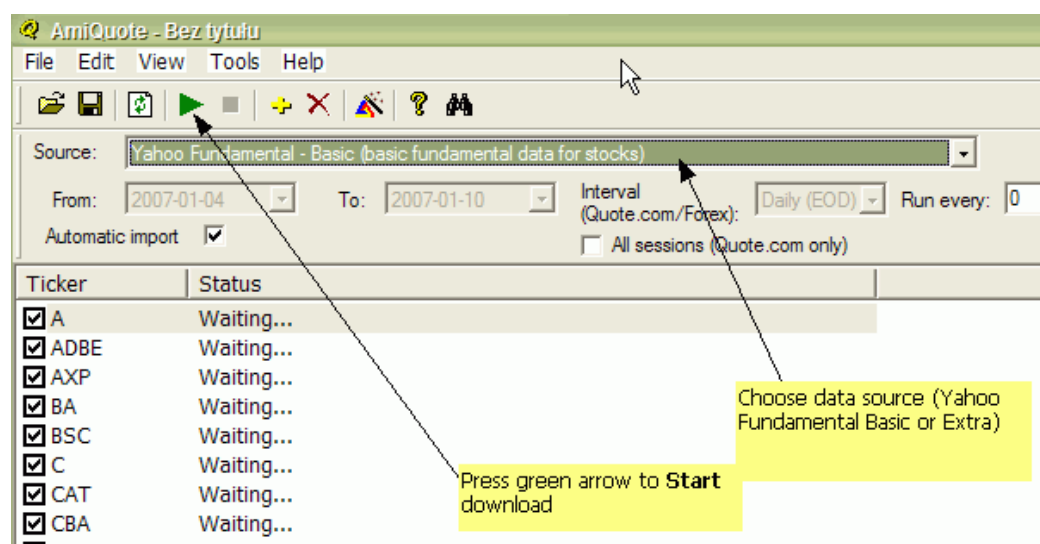
Dividend Date  
 Ex-Dividend Date  
 Last Split Factor  
 Last Split Date

Explanation of values: <http://help.yahoo.com/help/us/fin/research/research-12.html>

IMPORTANT NOTE: Unregistered version of AmiQuote allows you to download fundamental-ex data for first 20 tickers in the list. To download data for more symbols you need to register AmiQuote.

Downloading data is easy and straightforward:

1. Run AmiQuote
2. In AmiQuote, select **Tools->Get tickers from AmiBroker**
3. Select **Yahoo Fundamental – Basic** or **Yahoo Fundamental – Extra** from **Source** drop down list
4. Make sure that **Automatic import** box is checked
5. Press Green Arrow to **Start Download**



Once download is complete, you should see fundamental data updated in Information window in AmiBroker.

## ACCESSING FUNDAMENTAL DATA FROM FORMULA (AFL) LEVEL

To access fundamental data from AFL level you can use new [GetFnData](#) function. It has quite simple syntax:

```
GetFnData("field")
```

where "field" is any of the following fundamental data field supported. For detailed list please see [GetFnData function reference](#).

The function returns the number (scalar) representing current value of fundamental data item. There is no history of values (no arrays are returned), so it is useful for scanning, explorations (for current situation), market commentary / interpretation, but not for backtesting. Example exploration formula looks as follows:

```
AddColumn( Close / GetFnData( "EPS" ) , "Current P/E ratio" );
AddColumn( Close / GetFnData( "EPSEstNextYear" ) , "Est. Next Year P/E ratio" );
Filter = Status( "lastbarinrange" );
```

## IMPORTING FUNDAMENTAL DATA FROM OTHER SOURCES

AmiBroker allows also to import fundamentals using its flexible [ASCII importer](#) and/or [OLE interface](#) as all new fields are exposed as properties of [Stock](#) object.

ASCII importer \$FORMAT command now supports the following extra fields for fundamental data:

```
DIV_PAY_DATE
EX_DIV_DATE
LAST_SPLIT_DATE
LAST_SPLIT_RATIO
EPS
EPS_EST_CUR_YEAR
EPS_EST_NEXT_YEAR
EPS_EST_NEXT_QTR
FORWARD_EPS
PEG_RATIO
BOOK_VALUE (requires SHARES_OUT to be specified as well)
BOOK_VALUE_PER_SHARE
EBITDA
PRICE_TO_SALES (requires CLOSE to be specified as well)
PRICE_TO_EARNINGS (requires CLOSE to be specified as well)
PRICE_TO_BV (requires CLOSE to be specified as well)
FORWARD_PE (requires CLOSE to be specified as well)
REVENUE
SHARES_SHORT
DIVIDEND
ONE_YEAR_TARGET
MARKET_CAP (requires CLOSE to be specified as well – it is used to calculate shares outstanding)
SHARES_FLOAT
SHARES_OUT
PROFIT_MARGIN
OPERATING_MARGIN
RETURN_ON_ASSETS
RETURN_ON_EQUITY
QTRLY_REVENUE_GROWTH
GROSS_PROFIT
QTRLY_EARNINGS_GROWTH
INSIDER_HOLD_PERCENT
INSTIT_HOLD_PERCENT
SHARES_SHORT_PREV
FORWARD_DIV
OPERATING_CASH_FLOW
FREE_CASH_FLOW
BETA
```

Note that if you want to import only fundamental data with ASCII importer (without quotes) you need to use \$NOQUOTES 1 command. See Formats\laqfe.format and Formats\laqfn.format files for example usage – these

are files actually used by AmiQuote to implement automatic import of fundamental data downloaded from Yahoo.

The names of extra properties of Stock object are the same as used by [GetFnData](#) function and they are listed in detail in [OLE objects reference](#).

## How to get quotes from various markets

### REAL-TIME DATA (Professional Edition only)

Country/Exchange	Data source	Type	Price	Download	Update	Comments
<b>All US Stock and Futures markets.</b>  <b>FOREX</b>  <b>Major European markets.</b>	<a href="#">eSignal</a>	Real time streaming quotes.  Tick, 5-, 15-second 1-, 5-, 15-, 60-minute intraday  10-day tick, 60-day minute bar backfill.  Historical EOD (10 years)	eSignal Basic (equities) plan <b>from \$49*/month</b> +exch. fees. (50 symbols)  (*when paid yearly)  <a href="#">More pricing information</a>	Automatic	Automatic	<b>Dedicated RT plug-in – <a href="#">details here</a></b>
<b>All US Stock and Futures markets.</b>  <b>Major European markets.</b>	<a href="#">myTrack</a>	Real time streaming quotes.  1-, 5-, 15-, 60-minute intraday  15-day minute bar backfill.  Historical EOD (15 years)	myTrack RealTime Silver (\$19.95/month) or any other of <a href="#">service plans</a> plus SDK fee \$25/month	Automatic	Automatic	<b>Dedicated RT plug-in – <a href="#">details here</a></b>
<b>Various exchanges / various sources</b>	<a href="#">Quote Tracker</a>	Real time streaming quotes.	Various (including free)	Automatic	Automatic	<b>Dedicated RT plug-in – <a href="#">details here</a></b>

<b>(detailed list)</b>		1–, 5–, 15–, 60–minute intraday  Limited (max. 5 days, usually one day) backfills	<a href="#">More pricing information</a>			
<b>US stocks, futures, options, FOREX</b>	<a href="#">DTN IQFeed</a>	500 symbols, tick, 5–sec, 15–sec, 1–minute and up,  120 days backfill  (note: unfiltered feed)	\$50/month (includes FUTURES)	Automatic	Automatic	<b>Dedicated RT plug-in – <a href="#">details here</a></b>
<b>US stocks, futures, options</b>	<a href="#">QCharts/Quote.com</a>	(theoretically) unlimited symbols, 1 year of intraday backfill	\$95/month (QCharts basic)	Automatic	Automatic	<b>Dedicated RT plug-in available on request</b>  <a href="#">Click here to send e–mail for more details</a>
<b>Australian Stock Exchange</b>	<a href="#">MarketCast</a>	All ASX (satellite feed)	?	Automatic	Automatic	<b>Dedicated RT plug-in</b> – <a href="#">details here</a>
<b>US, Canada and European exchanges</b>	<a href="#">Interactive Brokers</a>	100 symbols streaming RT, 1–sec, 1–minute bars and up.  <b>30 day backfill available for IB customers</b>	\$10 per month in commissions, or free if your monthly commissions are >\$30	Automatic	Automatic	<b>Dedicated RT plug-in</b> – <a href="#">details here</a>
<b>Various</b>  (any data source that has DDE interface)	<a href="#">DDE link</a>	just streaming quotes, <b>no backfill</b>	Free	Automatic	Automatic	<b>Dedicated RT plug-in</b> – <a href="#">details here</a>

**END-OF-DAY, INTRADAY DELAYED DATA**

How to get quotes from various markets

AmiBroker can handle virtually EVERY exchange in the world if only plain ASCII data for that exchange are available. The table below list some of the data sources.

AmiBroker comes preloaded with sample DJIA components database. You can update this sample database (and any other US & Canada market databases) with a new quotes using supplied AmiQuote program.

Later in this tutorial you will find detailed instructions on [how to use AmiQuote](#).

**Quote sources for AmiBroker** (this list is not complete – keep in mind the fact that almost any source can be used). Use links to find out more (note that some links require internet connection)

Country/Exchange	Data source	Type	Price	Download	Update
<b>USA + Canada</b> (NYSE/Nasdaq/AMEX/TSE)	<a href="#">Yahoo Finance</a>	Historical + Current EOD	<b>Free</b>	Automatic ( <b>AmiQuote</b> )	Automatic
	<a href="#">MSN Money Central</a>	Historical EOD	<b>Free</b>	Automatic ( <b>AmiQuote</b> )	Automatic
	<a href="#">Quotes Plus</a> (recommended)	Historical + Current EOD + Fundamentals + Sectors/Industries	Paid	Automatic	Automatic
	<a href="#">TC 2000/TCNet</a> (stocks) <a href="#">TC2000 Mutual Funds</a>	Historical + Current EOD + Sectors/Industries	Paid	Automatic	Automatic
	<a href="#">FastTrack</a> (mutual funds)	Historical + Current EOD + Families	Paid	Automatic	Automatic
	FOREX	Historical EOD + Intraday	<b>Free</b>	Automatic ( <b>AmiQuote</b> )	Automatic
	<a href="#">CSI</a> <a href="http://www.csidata.com">http://www.csidata.com</a>	Historical EOD	Paid: <a href="#">Details here</a>	Automatic	Automatic
	<a href="#">Brite Futures</a>	Historical EOD (COMMODITIES ONLY)	<b>FREE</b>	Automatic	Automatic
	<a href="#">Quote.com</a> (US Stocks, Indices, Futures)	Historical EOD, Historical Intraday 1–min, 5–min, 15–min, 60–min	<b>Free*</b> (more info below) or PAID (Livecharts	Automatic ( <b>AmiQuote</b> )	Automatic

			basic)		
<b>Australia</b> (Australian Stock Exchange)	<a href="#">CoolTrader.Net</a>	Historical + Current EOD	Free	Manual	ASCII Import
	<a href="#">Norgate Investor Services</a>	Historical EOD (also US stock and future markets)	Paid	Automatic	Automatic (via MS plugin)
	<a href="#">BodhiFreeway</a>	Historical	Paid	Automatic (Bodhi downloader)	Automatic (via METASTOCK plugin)
	<a href="#">Yahoo Finance Australia</a>	Current EOD	Free	Automatic ( <b>AmiQuote</b> )	Automatic
<b>50+ International Exchanges</b>	<a href="#">Yahoo Finance</a>	Historical + Current EOD	Free	Automatic ( <b>AmiQuote</b> )	Automatic
<b>Poland</b> (Warsaw Stock Exchange)	<a href="#">Bossa.pl</a>	Historical + Current EOD	Free	Automatic (script-based)	Automatic (script-based)
<b>South Africa</b> (Johannesburg Stock Exchange)	<a href="#">Sharenet</a>	Historical + Current EOD	Paid	Automatic ( <b>Sharenet downloader</b> )	Automatic (script-based)
	<a href="#">Investor Data</a>	Historical	Paid	Manual	Manual
<b>Holland</b> (Amsterdam – Euronext)	<a href="#">PF-online</a>	Historical + Current EOD	Free	Manual	ASCII Import
<b>Other countries</b> ( )	<a href="#">DownloadQuotes.com</a>	Historical + Current EOD	Paid	Manual	ASCII Import

\* Please note that Lycos/Quote.com allows you to get FREE INTRADAY data without subscription, but you have to make sure that "USE LIVECHARTS ACCOUNT" is UNMARKED and "SERVER #1" is selected in the Tools->Settings page of AmiQuote.

## How to set up AmiBroker with eSignal feed (RT version only)

### Requirements

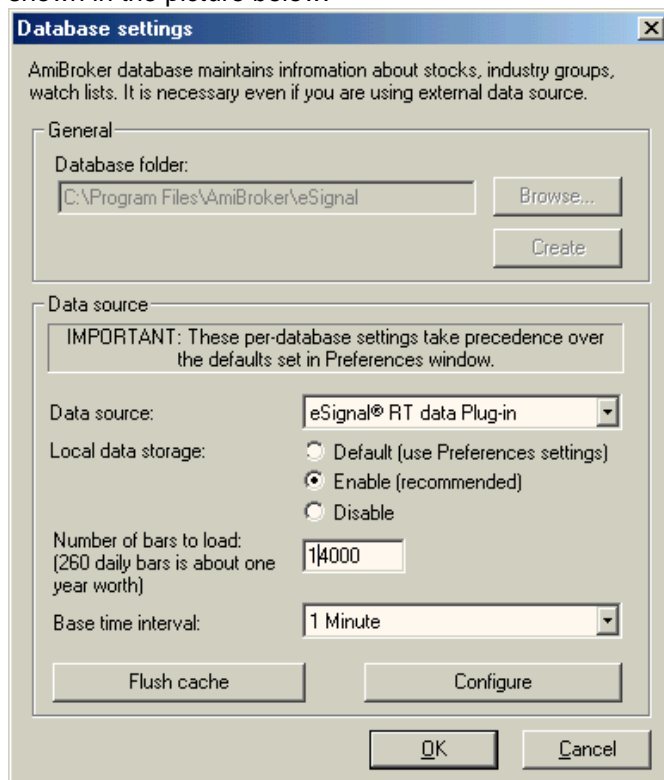
**IMPORTANT: You have to have eSignal application installed on your machine and a valid eSignal subscription.** [Click here](#) for special subscription discount.

### One-time setup

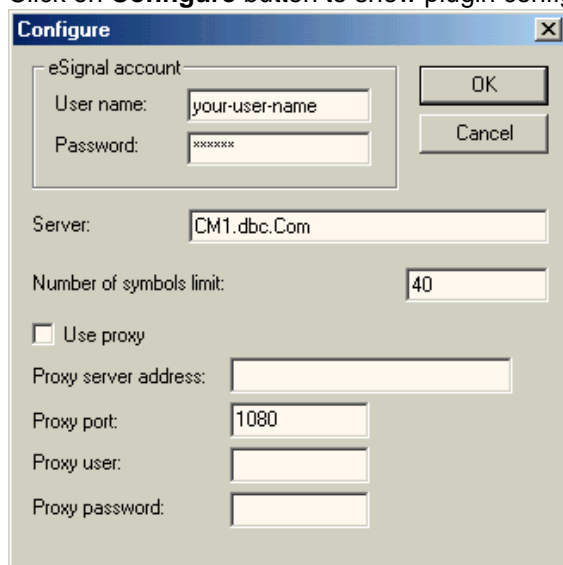
To use AmiBroker with eSignal feed you will need to perform a one-time setup described below:

- Run AmiBroker
- Choose **File->New database**

- Type a new folder name (for example: C:\Program Files\AmiBroker\Signal ) and click **Create** as shown in the picture below:



- Choose **eSignal RT data Plug-in** from Data source combo and **"Enable"** from **Local data storage**
- Enter appropriate **number of bars to load**:  
**90000 for 1-minute database combined with long history daily database**
- Click on **Configure** button to show plugin configuration dialog as shown below



Enter here your eSignal user and password (if you have eSignal properly installed AmiBroker will pre-set these fields to user/password entered in eSignal software). You may also adjust **Number of symbols**. This should not exceed your account limit and you may consider lowering this value if you want to use AmiBroker in parallel with another Data manager client application. (If you exceed the

limit of your subscription AmiBroker will re-adjust this number down)

Click OK

- Now choose Base periodicity. Note that recommended periodicity is 1 minute, but you can select all base periods starting from tick upto daily (End-of-day).

Note that selecting tick, 5-seconds or 15-seconds periodicities will cause transmission of huge amounts of data from eSignal servers (for actively traded security it can be several megabytes for just one symbol and very few days of history). If you have a modem connection this setting is highly discouraged. Also if you should consider using 5-second bars instead of pure ticks since this mode is faster.

If you want to have long daily histories AND intraday charts you should consider running TWO instances of AmiBroker. One for EOD charts and second for intraday charting. Both instances may use eSignal as a data source.

- Click OK.

From now on your AmiBroker reads quotes directly from the eSignal.

To learn how to use AmiBroker in Real Time mode read [this tutorial article](#).

## How to set up AmiBroker with myTrack feed (RT version only)

*Note: the most recent version of this document can be found at: <http://www.amibroker.com/mytrack.html>. Please check this page for updates.*

### Requirements

**IMPORTANT: You have to have myTrack subscription with SDK feature enabled.**

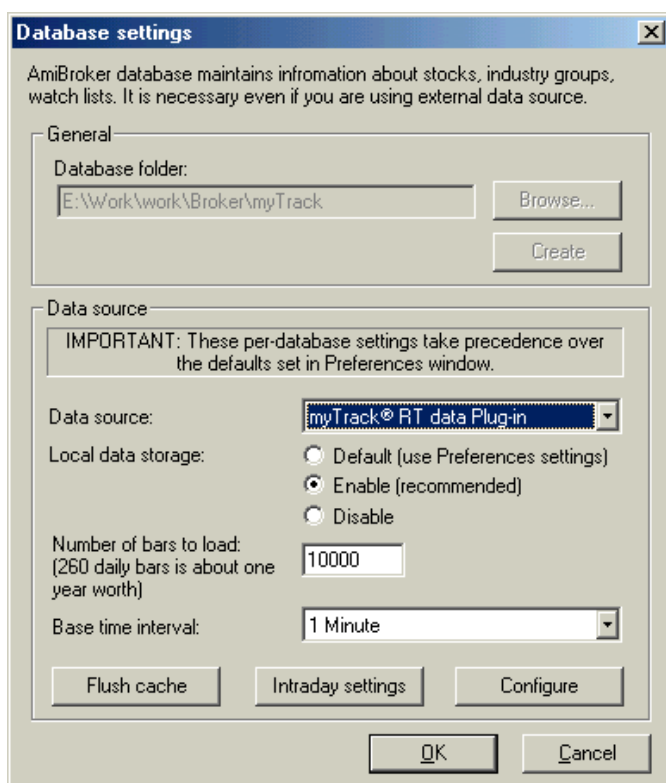
To have the SDK working, run the myTrack program, click on CHAT, then on Entitlements and then on Features, check the box SDK.

### One-time setup

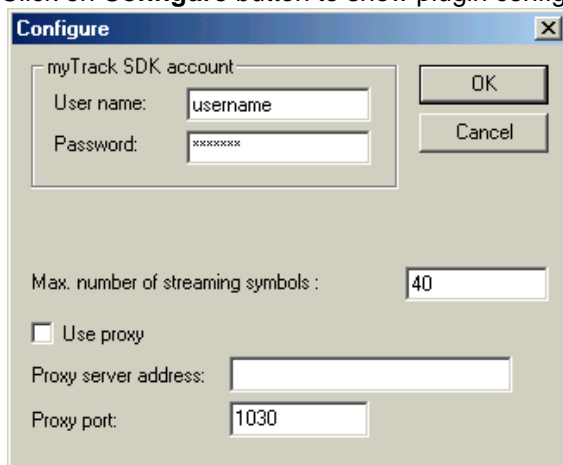
To use AmiBroker with myTrack feed you will need to perform a one-time setup described below:

- Run AmiBroker
- Choose **File->New database**
- Type a new folder name (for example: C:\Program Files\AmiBroker\myTrack ) and click **Create** as shown in the picture below:





- Choose **myTrack RT data Plug-in** from Data source combo and **"Enable"** from **Local data storage**
- Click on **Configure** button to show plugin configuration dialog as shown below



Enter here your myTrack user and password . You may also adjust **Number of symbols**. This should not exceed your account limit.

Click OK

- Now choose Base time interval. Note that supported bar intervals are **1 minute** and **daily (end-of-day)**.

If you want to have long daily histories AND intraday charts you should consider running TWO instances of AmiBroker. One for EOD charts and second for intraday charting. Both instances may use myTrack as a data source.

- Click OK.

From now on your AmiBroker reads quotes directly from the myTrack.

To learn how to use AmiBroker in Real Time mode read [this tutorial article](#).

## How to use AmiBroker with external data source (Quote Tracker)

IMPORTANT: You need QuoteTracker 2.4.9C OR ABOVE (3.1.0 recommended). Can operate on standard edition but AmiBroker RT is recommended.

**VERY IMPORTANT:** *QuoteTracker has to be CONFIGURED so its internal server is running. [Click here for the explanation](#).*

**CAVEAT:** QuoteTracker should be considered as poor-man's real-time substitute. Its performance can not be compared to true real-time feed as eSignal or myTrack that offer very reliable, long back-fills and true tick-by-tick updates.

**QuoteTracker plugin currently works in TWO modes:**

**daily mode** – plugin adds and updates the last (today's) bar with the most recent quotes in nearly real time– it means that you have to use it in conjunction with already existing end-of-day database.

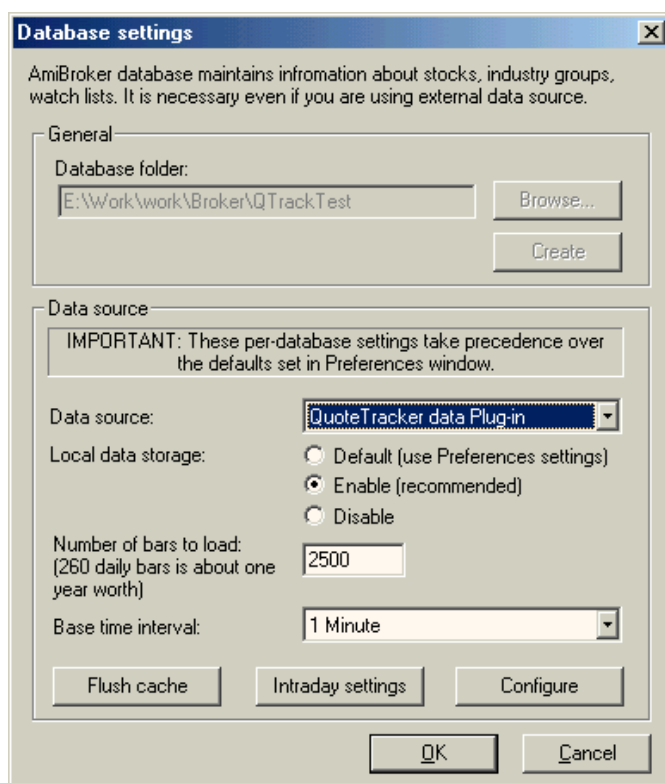
**intraday mode** – plugin provides one day intraday historical data – more days can be accumulated if AmiBroker with QT is launched everyday so AmiBroker can save histories to its local database.

### One-time setup

Make sure that your QuoteTracker has enabled QT HTTP server: **Options→Edit Preferences : Misc tab: HTTP Server Settings**  
**If you are using unregistered version of QuoteTracker make sure you click on ads often enough.**

To use an external data source with AmiBroker you will need to perform a one-time setup described below:

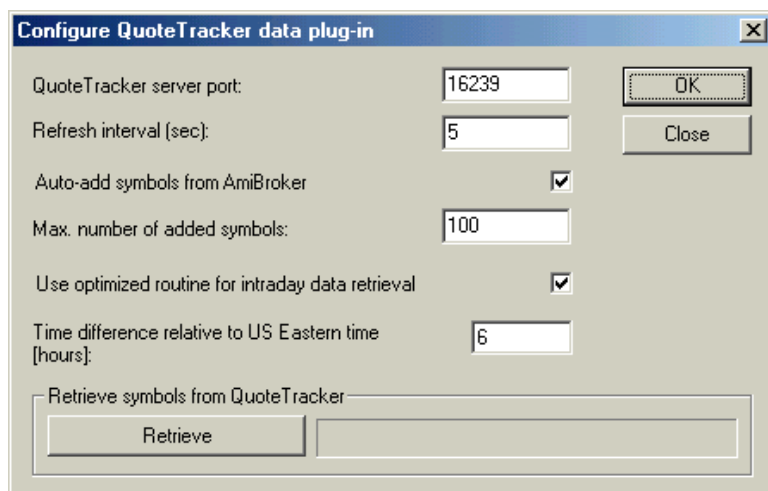
Run AmiBroker Choose **File→New database** Type a new folder name (for example: C:\Program Files\AmiBroker\NewData ) and click **Create** as shown in the picture below:



Choose appropriate entry from Data source combo:

- **Quote Tracker users** select "**Quote Tracker plug-in**" as a **Data Source** and "**Enable**" from **Local data storage**

Click on **Configure** button to show plugin configuration dialog as shown below

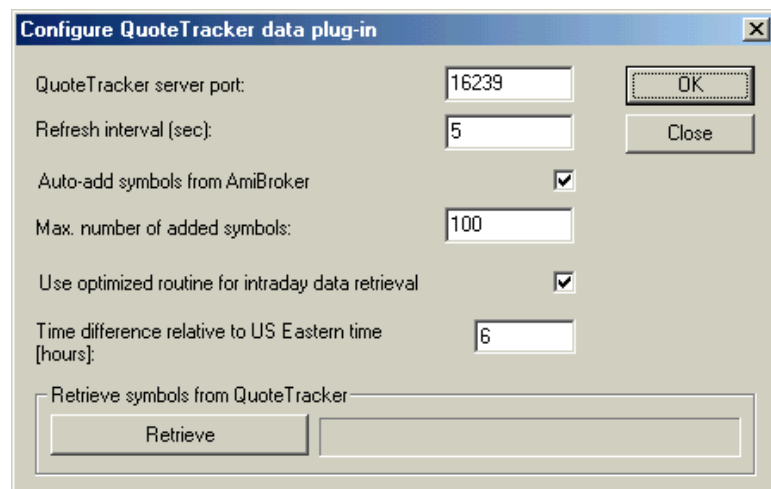


You may also click on **Retrieve** button to pre-fill AmiBroker database with symbols already present in QuoteTracker. From now on your AmiBroker reads quotes from Quote Tracker in nearly real time.

To learn how to use AmiBroker in Real Time mode read [this tutorial article](#).

### Description of QuoteTracker plugin configuration options

QT plugin configuration dialog looks as follows:



Here is a description of the settings:

**QuoteTracker server port:** defines the port on which QT HTTP internal server is visible. 16239 is the default value used by QuoteTracer and you should not change this in most cases. If in doubt please check QuoteTracker HTTP server settings: **Options->Edit Preferences : Misc tab: HTTP Server Settings menu of QT.**

**Refresh interval** – defines how often AmiBroker will ask QT for quotes. 5 second is default. You may consider changing it to 10 or 15 seconds in case you have lots of symbols and slow machine

**Auto-add symbols from AmiBroker** – if this option is turned ON (by default it is) if you switch in AmiBroker to the symbol that is not present in any of QT portfolios – it will be automatically added to default QT portfolio. It also applies to any other kind of access (for example if you try to import symbols to AmiBroker and they do not exist in QT – they will be added if this option is turned on). Switching it OFF disables auto-add feature.

**Max. number of added symbols** – defines the maximum number of symbols that get added using auto-add feature described above. This protects QuoteTracker from becoming overloaded (AmiBroker can handle tens of thousands symbols with ease but QuoteTracker can NOT)

**Use optimized routine for intraday data retrieval** – turning this on (default, recommended) significantly speeds-up data retrieval in intraday modes. If this option is enabled and AmiBroker already has partial intraday data for today AmiBroker asks QT just for a few last time and sales records that occurred since last update upto current time, if this option is disabled AmiBroker always asks QT for time&sales records from entire day.

**Time difference relative to US Eastern time** – the time difference (in hours) between your local time and US Eastern time (EST). This field is needed because QuoteTracker's server reports all times in EST time zone. This means that if you live in Australia QuoteTracker will report ASX quotes with EST time zone and they will be 15 hours off from your local time. While AmiBroker has the setting for shifting intraday charts and this is not a problem when running Intraday mode, it becomes a problem when using daily (EOD) mode because quotes

reported by QuoteTracker are one day off then. This setting solves this as AmiBroker adds the number of hours entered here to the time reported by QuoteTracker to get the valid date of quote in daily mode. This field is filled in with the difference calculated using your Windows Time settings.

**Retrieve symbols from QuoteTracker** – pressing "Retrieve" button adds all symbols present in QuoteTracker to AmiBroker symbol list.

## How to set up AmiBroker with IQFeed feed (RT version only)

*Note: the most recent version of this document can be found at: <http://www.amibroker.com/iqfeed.html> . Please check this page for updates.*

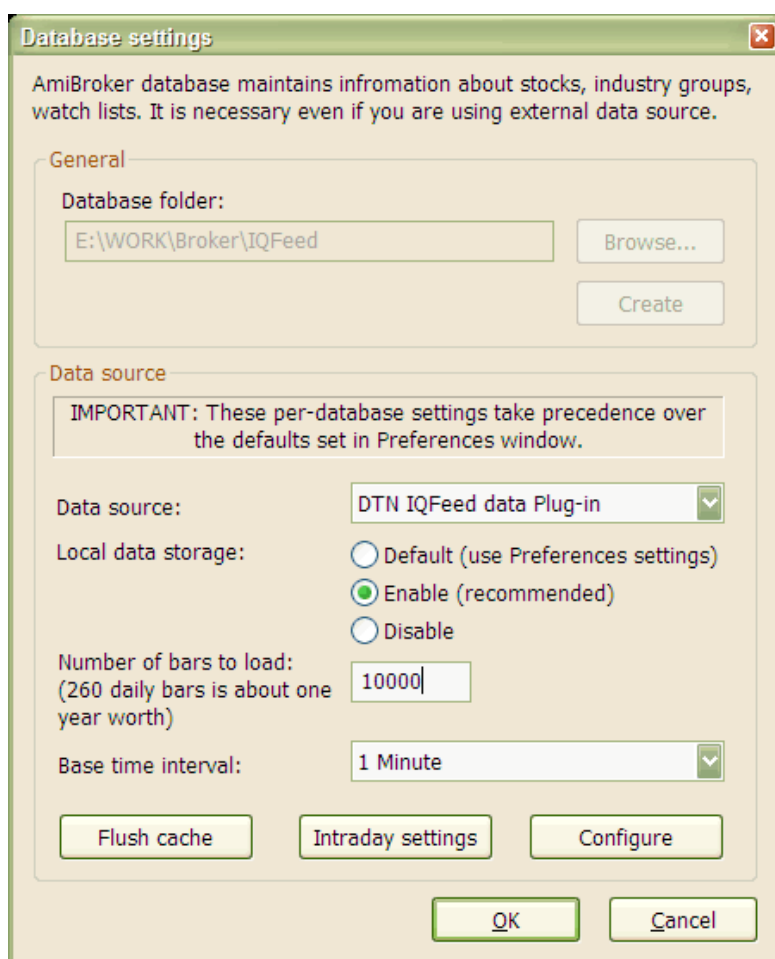
### Requirements

***If you don't have IQFeed CONNECTION MANAGER already installed you have to install it first. You can download [IQFeed client setup from here \(version 4.2.0.7\)](#).***

**<http://www.amibroker.com/video/IQFeed.html>**

To use AmiBroker with IQFeed you will need to perform a one-time setup described below:

- Run AmiBroker
- Choose **File→New database**
- Type a new folder name (for example: C:\Program Files\AmiBroker\IQFeed ) and click **Create** as shown in the picture below:



- Choose **DTN IQFeed data Plug-in** from Data source combo and **"Enable"** from **Local data storage**
- Now choose Base time interval. Select 1-minute
- Enter appropriate **number of bars to load**:  
**100000 for 1-minute database to get max history (8 months) available from IQFeed**
- **Click on "Intraday Settings". Check "Allow mixed EOD/Intraday data" box. Click OK**
- Click OK.

From now on your AmiBroker reads quotes directly from the IQFeed.

To learn how to use AmiBroker in Real Time mode read [this tutorial article](#).

## How to use AmiBroker with Interactive Brokers TWS

*Note: the most recent version of this document can be found at: <http://www.amibroker.com/ib.html> . Please check this page for updates.*

### IB PLUGIN FEATURES SUMMARY:

- supports upto 100 streaming symbols in real time (equal to IB TWS limit)
- supports all base time intervals: daily, hourly, 15-, 5-, 1-minute, 15-, 5-second, tick
- automatic connection (no need to manually "accept incoming connection" in TWS)
- supports upto **30 DAYS intraday data BACKFILL in 1-minute bar interval**
- upto 2000 bars backfill using 1-sec/5-sec/15-second bar intervals

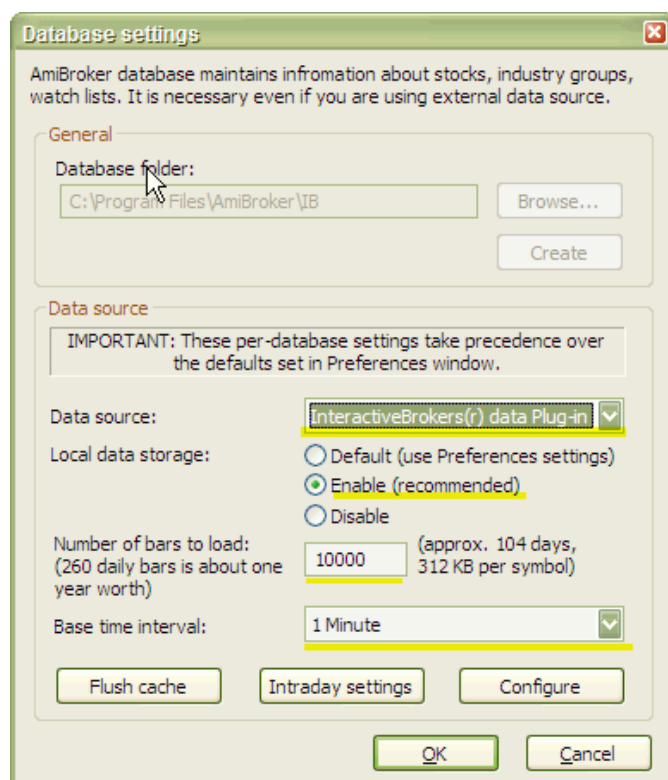
## INSTRUCTIONS:

*NOTE: Interactive Brokers TWS is CPU-hungry application, therefore for best results we recommend using machine with 1GHz processor or faster.*

*NOTE 2: There is a **VIDEO tutorial** showing how to set it up at <http://www.amibroker.com/video/ib.html>*

To use Interactive Brokers data plugin with AmiBroker you need to:

1. run [web-based TWS](#) or [download standalone TWS](#)
2. In TWS, select **Configure -> API -> Enable Active X and Socket clients**  
Also enter **127.0.0.1** in TWS, **Configure->API->Trusted IP addresses** menu to prevent "Allow incoming connection?" dialog.
3. Run AmiBroker and create new database with Interactive Brokers plugin as a data source, following these steps:
  - Run AmiBroker
  - Choose **File->New database**
  - Type a new folder name (for example: C:\Program Files\AmiBroker\IB ) and click **Create** as shown in the picture below:



- Choose **InteractiveBrokers(r) data Plug-in** from Data source combo and **"Enable"** from **Local data storage**
- Enter 30000 or more into **"Number of bars to load"** field
- Now choose Base time interval. **Supported intervals are: EOD, hourly, 15-minute, 5-minute, 1-minute. Professional Edition of AmiBroker allows also to select Tick, 5-second, 15-second intervals.**

Note that backfill is in bar interval of 1-minute or less (TWS limitation).

If you want to have long daily histories AND intraday charts you should consider running TWO instances of AmiBroker. One for EOD charts and second for intraday charting. Both instances may use IB as a data source.

- Click OK.

From now on your AmiBroker reads quotes directly from the Interactive Brokers.

---

## HOW TO USE BACKFILL FEATURE

Backfill feature in plugin 1.3.7 allows to download 24 intraday historical data to fill-in the gaps that may have occurred when AmiBroker / TWS is not running.

IB Backfill feature is configurable from **File->Database Settings, Configure :**

Configure Interactive Brokers plugin

Max. number of symbols : 100 OK Cancel

TWS host name or IP: 127.0.0.1

TWS socket port: 7496

Client ID: 33

**Backfill settings**

Base interval	Request length (in seconds)
1-minute	86400
15-second	30000
5-second	10000
1-second (tick)	2000

☐ Use RTH (TWS API 8.41 beta does not support it)

☒ Automatic backfill on first data access (causes additional traffic)

☐ Force instant quote retrieval for all symbols (causes more CPU usage)

Two main backfill-related settings are:

1. request length
2. automatic backfill



When request length is considered, as explained in TWS API Release Notes at:

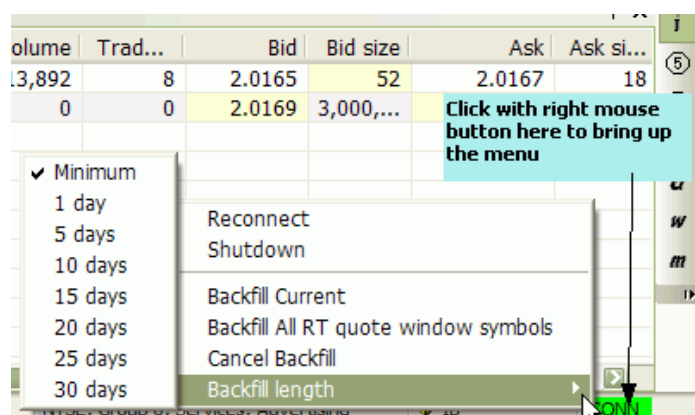
<http://www.interactivebrokers.com/en/software/apiReleaseNotes/apiBetanotes.php> currently IB backfill feature is limited to some fixed duration / bar interval ranges. For example you can get maximum 2000 1-second ticks, maximum 10000 seconds in 5-second interval (2000 bars), maximum 30000 seconds in 15-second interval (also 2000 bars) and **maximum of 5 DAYS of 1-minute bars**.

By default AmiBroker uses maximum allowable amounts.

As for "automatic backfill on first data access" – when it is checked AmiBroker attempts to backfill symbol when you display a chart for given symbol (or perform backtest or scan). Please note that TWS API currently allows only one backfill at a time so when there is a backfill already running in the background, automatic backfill request for next symbol will be ignored, until previous backfill is complete.

It is convenient to have this option turned on, however it can cause additional load on your internet connection because of data needed to be downloaded during backfill process.

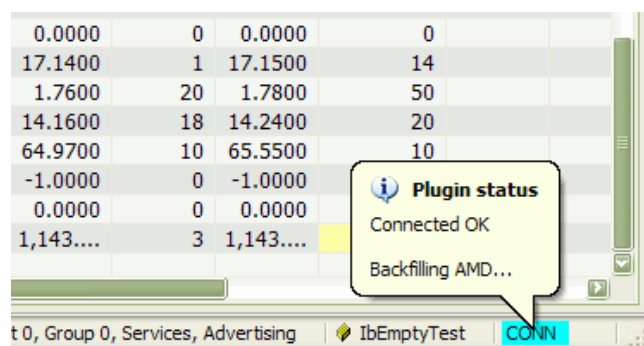
If you switch "automatic backfill on first data access" option off, you will still be able to backfill data for current symbol or all symbols in real-time quote window list using appropriate menu options from plugin status menu.



**Backfill Current** option allows to force backfill of currently selected symbol, while **Backfill All RT quote window symbols** allow to force backfill of all symbols listed in Real-Time Quote window. Backfill of multiple symbols is performed sequentially (one at a time) due to limitations of TWS.

**Backfill length** submenu allows to select desired backfill length.

During backfilling a tooltip pops up informing the user about symbol being currently backfilled and plugin status color changes to light blue (turquoise) as shown below:



## SYMBOLOLOGY

Symbol format now uses the symbol mode of TWS, not the underlying mode. The symbol mode in TWS can be seen in the '**View->Symbol Mode**' menu option in TWS.

The format is: *SYMBOL-EXCHANGE-TYPE*

where

*SYMBOL* is the same as the symbol column as displayed in TWS while under symbol mode

*EXCHANGE* (optional) is the exchange d in TWS while under symbol mode

*TYPE* (optional) is one the following:

STK – stocks, FUT – futures, FOP – options on futures, OPT – options, IND – indexes, CASH –cash (ideal FX)

Note that for stocks only the *EXCHANGE* and *TYPE* fields are optional. The exchange will be set to BEST (SMART) and the TYPE will be set to STK.

Please take special care when typing symbols as some of them (futures) have MULTIPLE SPACES in the symbol name. You have to type EXACTLY THE SAME number of spaces as provided in the examples below (see the dashes below symbol name that make it easier to see the number of characters)

Examples:

IB SYMBOL	Type	Description
CSCO	Stock	Cisco Corporation, Nasdaq
GE	Stock	General Electric, NYSE
VOD-LSE	Stock	VODAFONE GROUP, London Stock Exchange
ESM4-GLOBEX-FUT	Future	Emini ES Jun04 futures, Globex
QQQFJ-CBOE-OPT	Option	Jun 04, 36.0 CALL option QQQFJ
INDU-NYSE-IND	Index	Dow Jones Industrials Index
YM JUN 04-ECBOT-FUT --- -	Future	YM Jun 04 future, ECBOT (note 3 spaces between symbol and month and 1 space between month and year)
QMN5-NYMEX-FUT	Future	QM (Crude) June 2005 future contract, NYMEX
EUR.USD-IDEAL-CASH EUR.USD-IDEALPRO-CASH	Cash Forex	EURUSD currency pair, IDEAL EURUSD currency pair, IDEALPRO

Again:

ECBOT futures symbols have length of 21 characters with 3 spaces between contract symbol and month name and one space between month and 2 digit year

Contract	3 spaces	Month	spa	Year	-	E	C	B	O	T	-	F	U	T
----------	----------	-------	-----	------	---	---	---	---	---	---	---	---	---	---

							ce													
Z	B				J	U	N		0	4	-	E	C	B	O	T	-	F	U	T
Z	F				J	U	N		0	4	-	E	C	B	O	T	-	F	U	T
Z	N				J	U	N		0	4	-	E	C	B	O	T	-	F	U	T
Y	M				J	U	N		0	4	-	E	C	B	O	T	-	F	U	T

**NOTES ON IB API LIMITATIONS:**

1. **Backfill is available for REAL IB accounts only (not on demo)**
2. **Open** price is NOT provided by IB. For that reason Open field is empty in real time quote window
3. The data from IB does not include a timestamp on the trades. The current system time is used to timestamp each tick.
4. **IB TWS streaming data are NOT tick-by-tick, but rather 0.2–0.3 second snapshots, read this for details:** <http://www.interactivebrokers.com/cgi-bin/discus/board-auth.pl?file=/2/37364.html>

## How to use AmiBroker with external DDE data source

Note: the most recent version of this document can be found at: <http://www.amibroker.com/dde.html> . Please check this page for updates.

### WHAT IS DDE

DDE (Dynamic Data Exchange) is a Windows protocol used to allow applications to exchange data. For example, when you change a form in your database program or a data item in a spreadsheet program, they can be set up to also change these forms or items anywhere they occur in other programs you may use. DDE uses a client/server model in which the application requesting data is considered the client and the application providing data is considered the server.

Thousands of applications use DDE, including Microsoft's Excel, Word, Lotus 1–2–3, and Visual Basic.

For more information about DDE as communication mechanism in Windows please follow this link:

<http://msdn.microsoft.com/library/en-us/winui/WinUI/WindowsUserInterface/DataExchange/DynamicDataExchange/Ab>

### DDE FOR TRADERS

What DDE offers for traders? Basically real time streaming quotes. **There is NO BACKFILL via DDE.** Many real-time data providers and brokerages provide ability to get real-time data by means of DDE. You should ask your brokerage/real-time data vendor if they offer DDE link. The DDE plugin now available for AmiBroker allows to link to (almost) any DDE source (server) supplying real-time quotes. This makes it attractive option for all data sources that do not have dedicated plugin.

## WHEN NOT TO USE DDE PLUGIN

If you are using eSignal, IQFeed, Quote.com, MarketCast, and any other source that has dedicated plugin – you should use this dedicated plugin instead of DDE. This is so because dedicated plugins are ALWAYS better option (provide more features plus they are faster) than generic DDE.

## DDE PLUGIN FEATURES SUMMARY

- user–definable DDE server/topic/item for each field (open, high, low, close, volume, trade size, total volume, bid, bid size, ask, ask size, time)
- supports upto 500 streaming symbols in real time (version 1.1.0)
- supports all base time intervals: daily, hourly, 15–,5–,1–minute, 15–,5–second, tick
- NO BACKFILL (due to the fact that most DDE sources do not provide backfill)

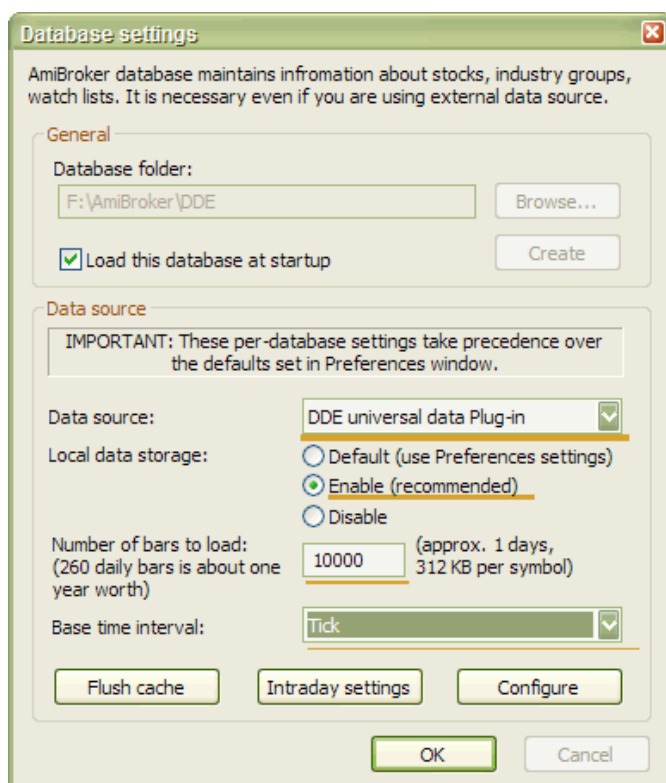
## HISTORY

- 1.2.2 – includes "Time shift" field in the context dialog, stores configuration per–database in dde.config file instead of in the registry plus other small improvements
- 1.2.1 – fixed problem with 'type mismatch'
- 1.2.0 – by default plugin uses regional settings numeric format now and CPU load is decreased
- 1.1.0 – symbol limit increased from 40 to 500
- 1.0.0 – initial release (BETA)

## INSTRUCTIONS

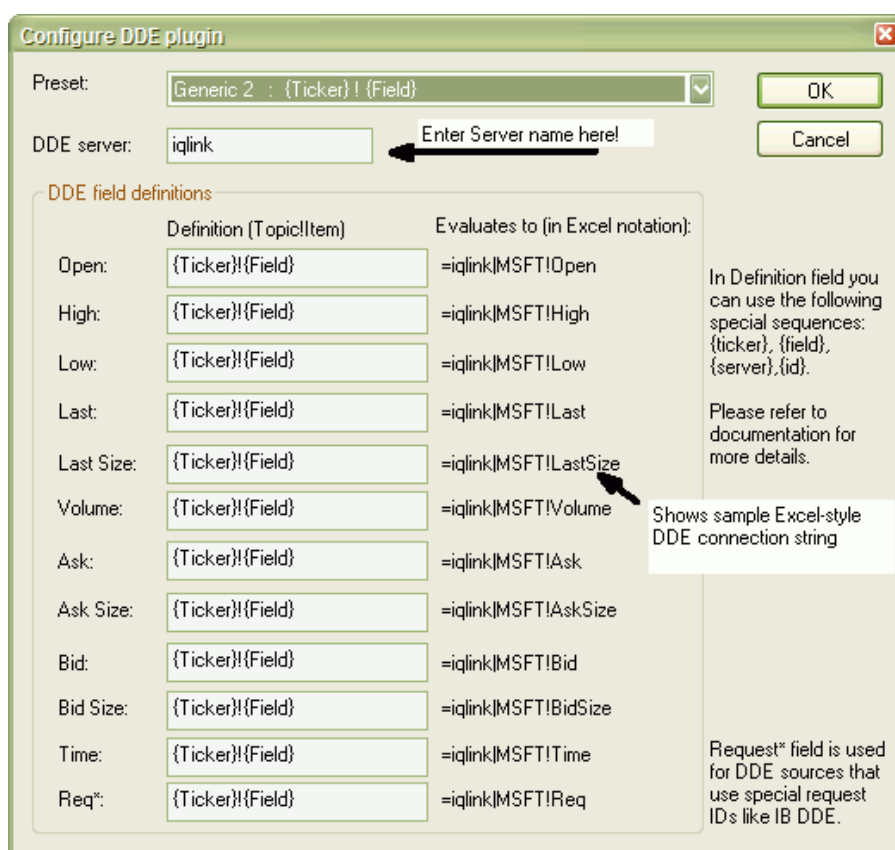
To use DDE data plugin with AmiBroker you need to:

1. (optional \*) Download the latest version DDE plugin from <http://www.amibroker.com/bin/DDE.dll> and copy it to PLUGINS subfolder of AmiBroker directory.  
\*Version 1.2.2 of DDE.DLL (Jun 7, 2007) is already included in AmiBroker 5.00 full setup
2. Enable DDE in the 3rd party software you are using as DDE server (consult data vendor/brokerage software documentation for details on how to enable DDE)
3. Run AmiBroker and create new database with "**DDE universal data plugin**" as a data source, following these steps:
  - Run AmiBroker
  - Choose **File→New database**
  - Type a new folder name (for example: C:\Program Files\AmiBroker\DDE ) and click **Create** as shown in the picture below:



- Choose **DDE universal data plugin** from Data source combo and "Enable" from Local data storage
- Enter 10000 or more into "**Number of bars to load**" field
- Now choose Base time interval. **Supported intervals are: EOD, hourly, 15-minute, 5-minute, 1-minute. Professional Edition of AmiBroker allows also to select Tick, 5-second, 15-second intervals.**
- Click **CONFIGURE** button – IMPORTANT: in the "CONFIGURE" dialog you have to setup all fields following the description of your data vendor.

Please check also paragraph below ("**CONFIGURING DDE PLUGIN TO WORK WITH YOUR VENDOR**") for detailed description. ATTENTION: you can not skip this part – without setting up fields specifically for your data vendor, the DDE WILL NOT WORK.



- Click OK.

The Plugin status indicator should change from Yellow "WAIT" to Green "OK" within a few seconds. If it does not turn to "OK" state it means that either:

- a) server name and/or fields are not set up correctly
- or
- b) DDE server (3rd party application) is not running or is not enabled

If indicator shows "OK" – then real time quotes flow into AB. You can check it by displaying View→Real time quote. Note: since there is no backfill you would need to wait for at least 3 bars of data to be collected before chart shows up.

## CONFIGURING DDE PLUGIN TO WORK WITH YOUR VENDOR

Various data vendors come use different DDE connection strings, here a few typical examples will be shown.

Most documentation of DDE uses Excel DDE syntax which looks as follows: **=SERVER/TOPIC!ITEM**

Server is a name of the DDE server such as WINROS, IQLINK, REUTER, CQGPC, MT, MTLINK, etc.

Topic is the topic of DDE conversation. Depending on Data source topic may be just the ticker symbol (like in IQFeed), or the field name (like in winros).

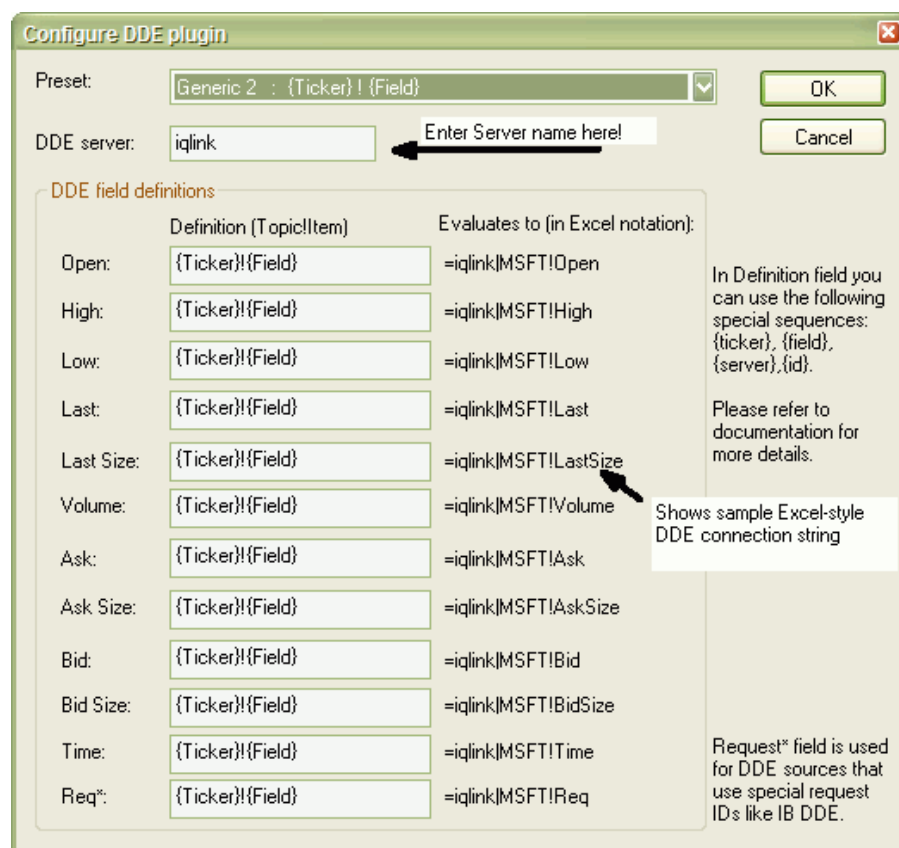
Item is the item of DDE conversation. Depending on data source it can be field name (like in IQFeed) or ticker symbol (like in Winros).

So DDE connection string in two most common standards look as follows:

=WINROS|LAST!MSFT

=IQLINK|MSFT!LAST

Now DDE plugin configuration screen looks like this:



In the UPPER part of the dialog you can see "DDE Server" field. In this field you should enter **SERVER** part of DDE connection string (**=SERVER|TOPIC!ITEM**) without equation mark and without | character.

Below you can see 12 text entry boxes where you can define DDE topic and item for each data field your data source provides. Here you should enter **TOPIC!ITEM** pair of the DDE connection string (**=SERVER|TOPIC!ITEM**) with exclamation mark between DDE topic and DDE item.

As you can see in the picture above, DDE plugin allow you to use a few special strings, namely: {Ticker}, {Field}, {FieldSp}, {Server}, {Id} which are evaluated in run-time for each symbol separately allowing to construct dynamic DDE strings (depending on selected ticker for example) required by most data sources:

- {Ticker} – evaluates to ticker symbol of given security
- {Field} – evaluates to the corresponding field name (without spaces), i.e. Open, High, Low, Last, LastSize, Volume, Ask, AskSize, Bid, BidSize, Time, Req
- {FieldSp} – similar to {field} but 2-word field names have spaces, namely: "Last Size", "Ask Size", "Bid Size"
- {Server} – evaluates to server name
- {Id} – evaluates to unique ID (running counter incremented by 1 with each symbol)

All other texts are carbon-copied, so if you write for example:  
PREFIX\_{Ticker}\_SUFFIX!MYTEXT

it will evaluate to =SERVER|PREFIX\_MSFT\_SUFFIX!MYTEXT (provided that current symbol is MSFT)

Next to field definitions we can see what given definition will evaluate to (in Excel notation). This makes it easy to verify if definition is correct.

Sample evaluation uses always "MSFT" as a {Ticker}, and 34 as {id}.

If your data source does not provide all fields you can make given field empty. Note that for proper operation the "Last" price (the price of last trade) is required. If your data source does not provide "last" price (most of forex sources don't have "last") you can force DDE plugin to use "Bid" instead. For that you should make "Last" field blank and provide appropriate DDE topic!item pair in "Bid" field. Please also note that Topic!Item pairs should evaluate to unique values.

In the top part of the dialog you can see "Preset" combo-box.

As of now it allows to pre-set the fields using two generic schemes:

- a) {Field}!{Ticker} – "last price" evaluates to =SERVER|Last!MSFT
- b) {Ticker}!{Field} – "last price" evaluates to =SERVER|MSFT!Last

In the future "Preset" box will contain more presets for various DDE source that you submit.

## A FEW EXAMPLES

Connection examples are shown on the web page: <http://www.amibroker.com/dde.html>

## TEST PLATFORMS

DDE plugin has been tested and it is known to work properly on Windows XP (32 bit DDE) and Windows 9x (16 bit DDE). The following DDE servers are verified by us to work properly:

- IQLINK (DTN)
- WINROS (eSignal)
- MT (Metaquote)

DDE plugin does NOT work with the following DDE servers:

- VTSPOT (Visual Trader) – due to improper coding in VisualTrader that causes Microsoft DDEML library DdeConnect function to hang on the very first connection attempt

All other DDE servers not listed above should work properly. Contact support at [amibroker.com](http://amibroker.com) in case of problems.

## HELP US TO HELP THE OTHERS:

In order to help the others to configure DDE plugin for their data vendor, once you succeeded to link with your particular vendor please drop as a note with a screenshot of the CONFIGURE dialog and name of the source. This will be later included in this document as a reference how to use various data sources. Also working setups will be added to "presets" combo for easy one-click configuration.



**NOTES ON DDE PLUGIN:**

1. There is **NO BACKFILL** in DDE plugin. You can use however ASCII importer (this includes AmiQuote) to import historical data right into the database that you will update later in real time using DDE plugin.
2. **Change, % change** fields are NOT available (yet)
3. **Time** and **Req** fields are now ignored (this may change in the future)
4. The current system time is used to timestamp each tick.
5. When your source does not offer "LAST" price (like several Forex sources) you should make "Last" field EMPTY in the configuration dialog. This will tell the plugin to use "BID" field instead.
6. Plugin status (connected/disconnected) always initially comes up with "Wait" state (Yellow indicator). It means that no DDE conversation has been established. If at least ONE DDE conversation starts successfully it will turn to "OK" state (green indicator). If DDE server was not running at first attempt to connect, the plugin will NOT attempt to reconnect automatically. Instead you should force reconnection manually (see point 7). The indicator may turn to "Disconnected" (red indicator) only in two cases:
  - a) you were connected properly but DDE server (3rd party app) has been closed
  - b) you selected "shutdown" from plugin status menu
7. You can reconnect at any time by selecting "reconnect" from plugin status menu.

## How to work with Real-Time data plugins

### One-time setup

In order to use AmiBroker with any real-time data source you have to set up the database with appropriate data plug-in first. This is required only once at the database creation time. Instructions for setting up are available here: [eSignal](#), [myTrack](#), [IQFeed](#), [QuoteTracker](#), [MarketCast](#).

Check also on-line data sources page at <http://www.amibroker.com/quotes.html> for new plugins.

### Adding symbols

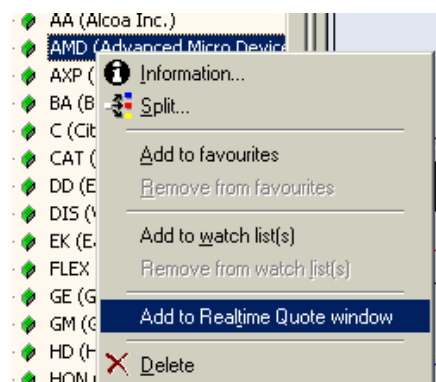
Now you can add symbols to your database. To do so please go to **Symbol->New** menu. In the add symbol dialog enter one or more tickers (comma separated) you wish to add to your database. If you want to see the chart for newly added symbol just select it from the Symbol tree in the workspace window. Please allow few seconds (depending on the speed of your internet connection) to backfill historical data.

You may add more tickers that your RT account allows. AmiBroker will automatically switch/update/refresh symbols so the most recently used symbols are active and older ones are automatically removed from Data manager. Doing so however may lead to some problems if you exceed your subscription limits too much. So it is advised to use this feature responsibly and not expecting getting 500 symbols while your subscription is limited to only 50.

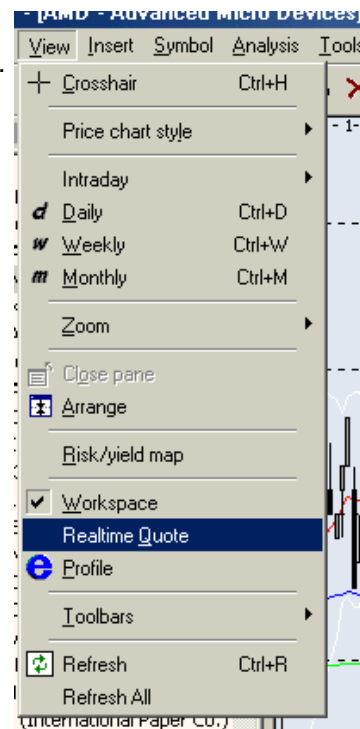
Note that the above mechanism does not apply to real time quote window and it can not hold more symbols than your subscription limit.

## Showing real time quote window

AmiBroker RT features real-time watch window that allows you to watch streaming quotes. To show this window choose **View->Realtime Quote** menu. (see image to the right ---->)

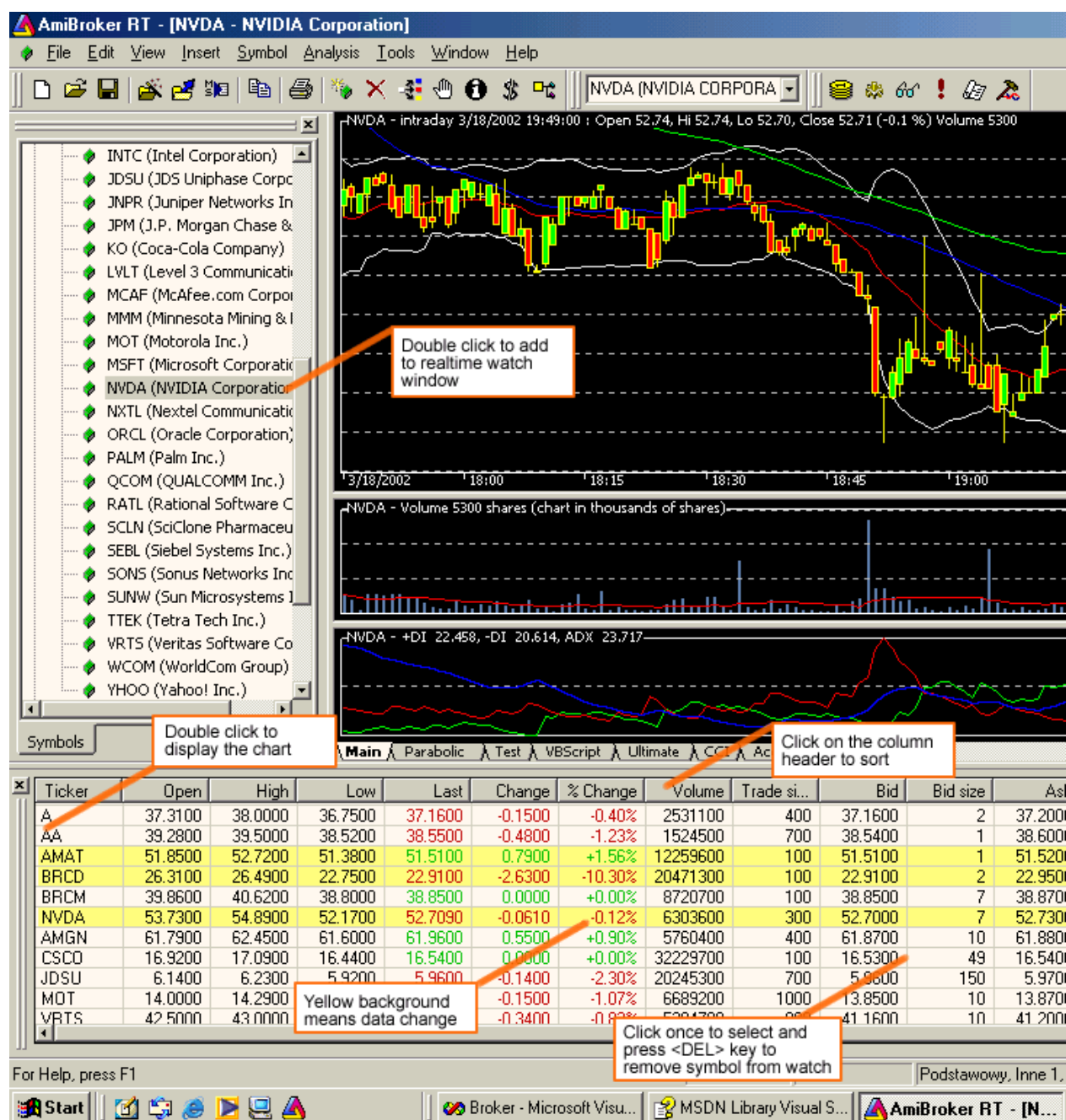


To add symbols to Realtime quote window you either double click on the symbol tree or use right mouse button menu Add to Realtime quote option as shown in the picture above.



## Working with real time quote window

The RT quote window provides real-time streaming quotes and some basic fundamental data. It is fairly easy to operate as shown in the picture below:



You can also display context menu by pressing RIGHT mouse button over RT quote window.

Realtime Quote							
Ticker	Open	High	Low	Last	Change	% Change	Volum
AAPL	68.2950	71.4300	68.2000	71.1700	3.9600	+5.89%	60,459,82
ADBE	35.9400	36.1000	35.6000	36.0000	0.0100	+0.03%	1,860,12
ALTR	20.8200	21.2600	20.7500	20.9100	0.0400	+0.19%	2,722,91
AMAT	18.3200	18.6000	18.2600	18.4700	0.2100	+1.15%	12,447,14
AMGN	72.5600	73.1200	71.5700	71.7200	-1.1400	-1.56%	5,411,84
AMZN	37.2000	38.4300	37.1300	38.3500	1.0100	+2.70%	4,251,95
APCC	23.0200			23.0100	0.0300	+0.13%	588,46
APOL	52.9500			52.7100	1.5300	+2.99%	2,546,04
ATYT	16.9400			17.0800	0.1400	+0.83%	5,244,00
BBBY	40.8400			40.7400	2.4200	+6.32%	11,001,66
BEAS	13.3800			12.9800	-0.4200	-3.13%	5,923,80
BIBB	45.3000			44.7775	-0.5825	-1.28%	1,382,83
BMET	37.7500			38.0000	0.2200	+0.58%	1,622,57
BRCM	45.8500			46.2300	0.1760	+0.38%	8,992,08
CDWC	58.7500			58.0630	-0.7070	-1.20%	362,83
CECO	40.8300			40.7200	-0.9800	-2.35%	1,708,99

The context menu allows you to access the following options:

- **Time & Sales**  
Opens [Time & Sales window](#) that provides information about every bid, ask and trade streaming from the market.
- **Easy Alerts**  
Opens [Easy Alerts window](#) that provides way to define realtime alerts executed when bid/ask/last and other fields hit user-defined levels
- **Add Symbol**  
Adds current symbol to Real-Time Quote list
- **Add watch list...**  
Adds entire watch list to real-time quote window
- **Remove Symbol**  
Removes highlighted line (symbol) from the Real-Time Quote list.
- **Remove All**  
Removes all symbols from real-time quote list
- **Hide**  
Hides Real-Time Quote list

## Working with intraday and daily charts

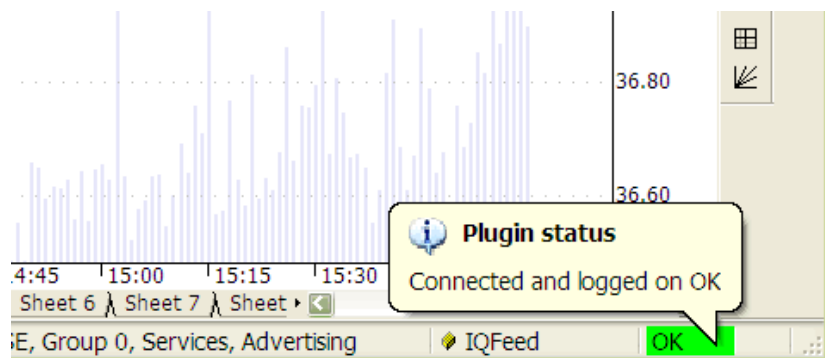
If your data source supports mixed EOD/Intraday mode (such as eSignal or Marketcast) you can use single database for both types of charts.

However if your data source does not support mixed EOD/Intraday mode and if you want to have long daily histories AND intraday charts you should consider running TWO instances of AmiBroker. One for EOD charts and second for intraday charting. Both may use the same real-time data source.

## Connection status display

The data plug-in connection status is displayed in the plugin status display area located in the lower right part of the AmiBroker main window as shown in the picture below. When connection status changes AmiBroker

plays a beep sound and pops up bubble tool tip to inform about status change.



The bubble tip provides more detailed information text and disappears automatically after 2 seconds.

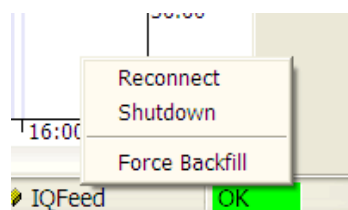
If you want to re-display it just hover your mouse over plugin status display area.

To enable quick examination of connection status AmiBroker displays color coded information:

- **OK (green light)** means that connection is OK and indicates correct operation of the plugin
- **WAIT (yellow light)** means that connection is being set-up right now or the plugin is connected only partially (to few of many servers). Usually this state is transient and within few seconds the status comes back to "OK".
- **ERR (red light)** means that connection is broken. It may mean invalid user name/password for your subscription, or the fact that some 3rd party component / program required is not running (for example if QuoteTracker is not running and you are using QuoteTracker plugin). This state usually requires some user intervention such as checking/fixing user/password in File->Database Settings->Configure or running required component. When you fix the reason the plugin will automatically attempt to reconnect (and if reconnect is successful then "OK" will be displayed)
- **SHUT (purple light)** means that some serious problem occurred and the plugin will not attempt to reconnect automatically. In most cases you have to first fix the problem that caused this state and then reconnect manually using plugin context menu described below. Alternatively you can just restart AmiBroker.

### Using plugin context menu

Real time plugins provide some additional controls via plugin context menu. This context menu is available when you click with RIGHT mouse button over plugin status display area. If you do, the menu like this will be displayed:



Please note that various plugins offer various options in this menu, however most plugins provide at least 3 basic and useful options:

- **Reconnect** – this option allows you to reconnect manually. Most RT plugins attempt to reconnect

automatically, but sometimes manual reconnect is necessary.

- **Shutdown** (Disconnect) – this allows to shutdown RT plugin. This is useful when you want to stop streaming of quotes.
- **Force backfill** – this option causes that plugin re-downloads entire (intraday) history from the server. Usually the plugin automatically handles all backfills so you don't need to trigger backfills by hand. If the plugin detects that you have some missing quotes from last available bar till current date/time it triggers backfills and it is all automatic. But... in at least two cases this option is useful:
  - ◆ backfilling more bars after settings change (when you enlarge 'number of bars to load' in File->Database Settings dialog you have to force backfill for symbols that were backfilled previously with smaller number of bars)
  - ◆ cleaning up bad ticks (when you see a bad tick you may try forcing backfill in hope that data vendor has cleaned up its database and you will get fixed data – works well for eSignal that really appears to fix bad ticks after they happen)

### Things you should NOT do, or you should do very carefully

You should note the fact that when you are using data plugin then the plugin controls the quotation database (see [Understanding database concepts](#) article), therefore you should NOT import quotes from ASCII files (this includes AmiQuote) for symbols that are already present in the real-time database.

If you do, the plugin will eventually overwrite your imports with the real-time data or your database will become corrupted (if you import end-of-day data over intraday database).

So please do not import ASCII (especially EOD data) into real-time intraday database fed by the plugin.

You may ask: why this is not disabled at all. The answer is that sometimes it is useful and sometimes it will work (but these are rare cases). For example it will work if you import INTRADAY data into the intraday database fed by QuoteTracker plugin and both the database and imported data have exactly the same bar interval.

It also works if you import the data for symbols that are NOT present in the database. In this case newly imported symbols are marked by ASCII importer as "use only local database for this symbol" (See [Information window](#) for details), so they are EXCLUDED from the real-time update. This is useful if you want to import some other data (even not quote data) and access it via [Foreign](#) function while using your real-time database.

So ASCII import is not disabled in real-time database but you have to use it with extreme care and know what you are doing.

Second thing is using [Quote Editor](#). Although data are controlled by the plugin it is in most cases possible to use Quote Editor. However please note that you will be able to edit only 1-minute data or higher interval, and you will be able only to edit symbols that are backfilled completely (there is no running backfill for the particular symbol) and you will NOT be able to edit last three bars. This is so because last three bars are cached in the plugin. So you will be able to edit them only when new bars arrive making them 'older' than last three.

### 'WAIT FOR BACKFILL' feature

The users of eSignal, myTrack and IQFeed real-time plugins may now check "wait for backfill" box in the Automatic analysis window and all scans, explorations and backfills will wait for completion of backfill process for given symbol. This flag has no effect on databases that do not use plugins (external data sources) or use end-of-day plugins (like FastTrack, QP2, TC2000/TCNet, etc). This flag has also no effect when using QT plugin due to the fact that QuoteTracker manages backfills by itself and does not provide any control of

backfill process to 3rd party applications.

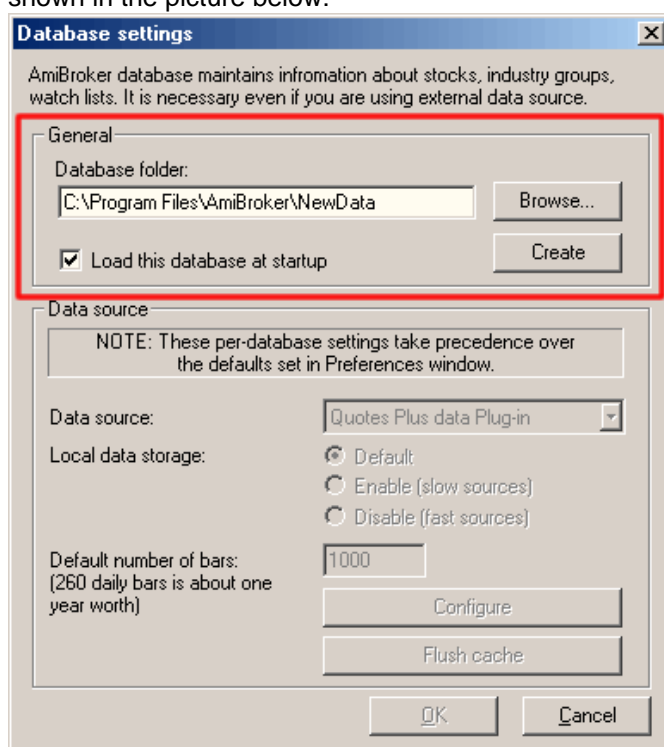
## How to use AmiBroker with external data source (Quotes Plus, TC2000/TCNet/TC2005, FastTrack, Metastock)

One of the new features introduced in AmiBroker version 3.90 is the ability to read directly external databases. This is achieved by means of data plug-in DLLs that allow to link AmiBroker database with an external source. Please note that although you will be using external database, you will still need an AmiBroker database for storing additional information that is not supported by the external source like hand-drawn studies, assignments to groups, watch lists, composites and so on. You can find more information on AmiBroker database handling [here](#).

### One-time setup

To use an external data source with AmiBroker you will need to perform a one-time setup described below:

- Run AmiBroker
- Choose **File->New database**
- Type a new folder name (for example: C:\Program Files\AmiBroker\NewData ) and click **Create** as shown in the picture below:



- Choose appropriate entry from Data source combo:
  - ♦ **Quotes Plus users** select "Quotes Plus plug-in" as a **Data Source** and "Disable" from **Local data storage**
  - ♦ **TC2000/TCNet users** select "TC2000/TCNet plug-in" as a **Data Source** and "Enable" from **Local data storage**
  - ♦ **TC2000 for Mutual Funds users** select "TC2000 Mutual Funds plug-in" as a **Data Source** and "Enable" from **Local data storage**



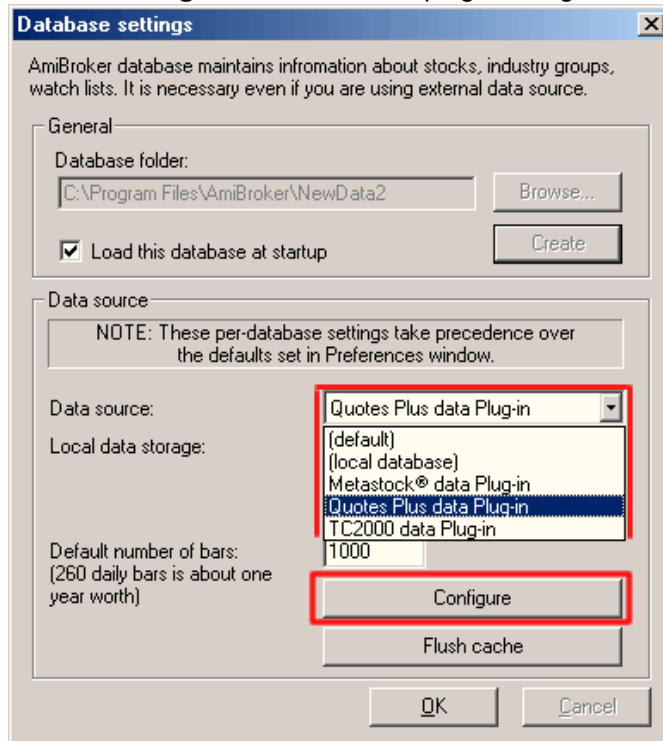
- ◆ **TC2005 users** select "TC2000/TCNet plug-in" as a **Data Source** and "Enable" from **Local data storage**

Note: TC2005 users may need to [follow these instructions \(click here\)](#) if TC2000 plugin does not show up.

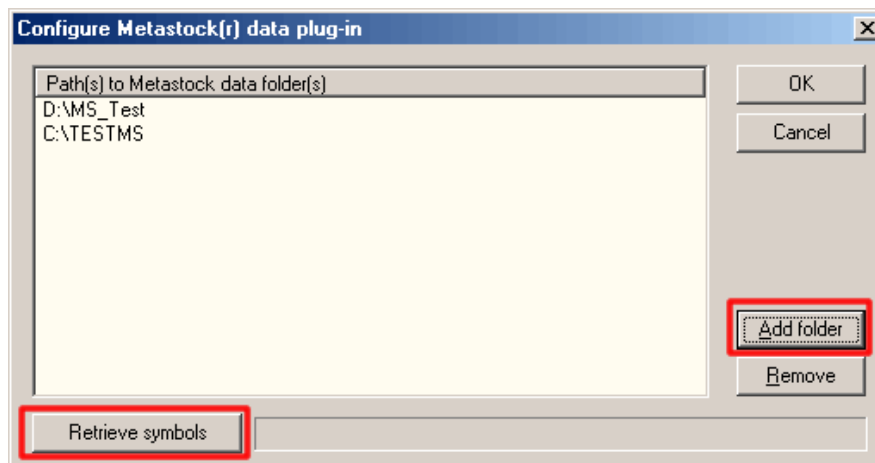
- ◆ **FastTrack users** select "FastTrack plug-in" as a **Data Source** and "Disabled" from **Local data storage**

- ◆ **Metastock users** select "Metastock plug-in" as a **Data Source** and "Disable" from **Local data storage**

- Click on **Configure** button to show plugin configuration dialog as shown below



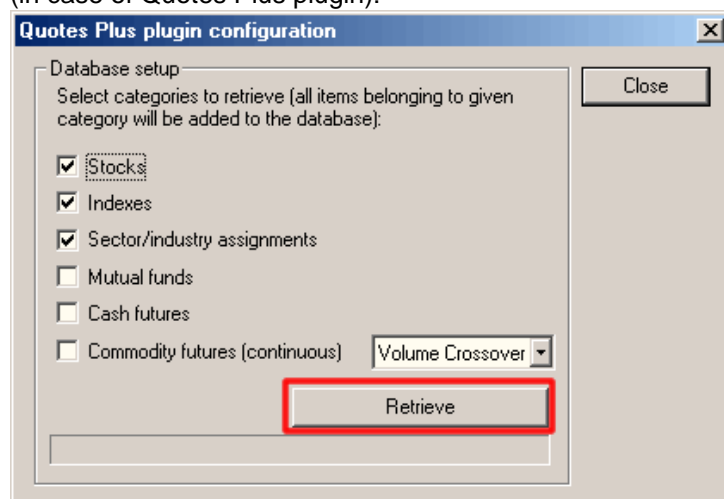
- **Metastock plug-in only** (skip this point in case of TC2000, Quotes Plus, FastTrack):  
Click on the "Add folder" button to add Metastock database directory as your data source (browse for Metastock MASTER file and click OK) as shown below:





– you can add unlimited number of Metastock directories effectively overcoming MS 4096 symbols limitation.

- Click **Retrieve** button – this will setup a new database with all symbols and full names. Quotes Plus and TC2000 plugins will also setup your sectors/industries names and assignments, as shown below (in case of Quotes Plus plugin):



From now on your AmiBroker reads quotes directly from the external data source. No need to import/update quotes anymore. **All new quotes will appear automatically without user intervention.**

IMPORTANT: If there are **new symbols** added or **old symbols** deleted to/from the external data source, you will need to go to File→Database Settings→Configure and click "RETRIEVE" again to get new symbols.

### Plug in performance notes

Using AmiBroker native database gives absolutely the best performance (it takes less than 2 milliseconds to retrieve 1000 data bars).

Metastock plugin is also quite fast, as it can retrieve 1000 bars in about 6–7 milliseconds (including looking up for symbol in 5 different directories). In fact AmiBroker can access Metastock data faster than Metastock itself :-)

Quotes Plus performance depends on various factors – first access can be much slower (0.1–0.2 sec for 1000 bars) but subsequent accesses are faster (down to 5 milliseconds). FastTrack plugin is as fast as Quotes Plus plugin.

TC2000 is not as fast, especially if you are using data only on CD. So it is advised to copy your database to hard disk for better performance. But still, even when using CD-only data, AmiBroker can access 1000 bars from TC2000 in about 0.25 sec (first access) and 0.015 sec (subsequent accesses). Also it is advised to enable "**Local data storage**" when using TC2000 plugin because it gives tremendous (>10 times) speed-up (once you access the TC2000 data, AmiBroker caches them in its own native database for fast retrieval).

Times are approximate and do not include one-time plug-in initialization process. Measurements were done on fairly low-end Celeron 600 based computer with 196KB RAM and 24x CD-ROM

### In-memory caching

By default AmiBroker holds only 10 the most recently accessed symbols' data in RAM. This takes up about 320 KB (yes, kilobytes) of memory for 1000 bars per symbol loaded. You can enlarge "**In memory cache**" (**Tools→Preferences : "Data"** tab) to 100 (approx. 3.2MB additional RAM consumption) or 1000 (approx.

32MB additional RAM consumption) or even more to get much better performance for subsequent data access (once data are in RAM AmiBroker does not need to ask plugin again and again)

## How to update US quotes automatically using AmiQuote

### QUICK START

Run AmiBroker

Choose **Tools→Auto-update quotes (US & Canada)**

### HOW IT WORKS

AmiQuote loads (or retrieves from AmiBroker) a ticker list file (.TLS) which is simple ASCII file with ticker symbols, then parses it and generates URLs to the Yahoo! finance site based on ticker name, mode (current quotes or historical), country and From/To date. Then, when you start the download process, it requests the data from Yahoo and stores downloaded data in the separate .AQD (daily) or .AQH (historical) files for each ticker. After download, if AmiBroker is running, AmiQuote will import the quotes into AmiBroker automatically.

### USAGE

#### Automatic update

The easiest method to work with AmiQuote and AmiBroker is to use the procedure given in **Quick Start** section of this document. Just run AmiBroker and AmiQuote and choose **Tools→Auto-update AmiBroker database**. This method updates historical quotes from the last date present in AmiBroker upto today. When performing automatic update, AmiQuote performs internally 4 steps a) retrieves the ticker list from AmiBroker (all symbols loaded currently in AmiBroker); b) gets the last quotation date available in AmiBroker; c) performs historical download from last date upto today; d) instructs AmiBroker to import downloaded files.

Please note that this procedure works only for US & Canada markets, because Yahoo provides historical quotes only for that markets.

Note that AmiQuote currently supports a new command line parameter: **/autoupdate**. This option forces AmiQuote to perform automatic update procedure without user intervention. By default AmiBroker's Tools menu is configured as follows:

```
C:\Program Files\AmiBroker\AmiQuote\Quote.EXE /autoupdate
```

So, you are able to update your US database with a single click on **Tools→Auto-update quotes (US & Canada)** in AmiBroker

#### Manual operation

Automatic mode is nice but there are cases when you have to perform some tasks manually. There is a good old document describing that mode of operation at: [How to download quotes manually using AmiQuote](#) . Everything written in this document remains valid with one exception – now importing to AmiBroker are performed automatically if you have **Automatic import** checkbox marked.

There are also several cases when you prefer to do things manually, then please don't forget about some useful tools available at your fingertips:

**File→Open, File→Save, File→Save As**

These functions enable you to load and save your edited ticker lists for future repeated use.

**Edit→Add tickers**

This function allows you to add the tickers to the list. Just type space separated tickers into the field that will show up when you choose this function.

**Edit→Delete tickers**

This function allows you to delete tickers from the list. Just select the items you want to delete from the list view (multiple selection possible by holding SHIFT or CTRL key while clicking on items), and choose this function.

**Edit→Mark all, Edit→Unmark all, Edit→Toggle, Edit→Mark selection, Edit→Unmark selection**

These functions allow you to mark the tickers for download. Please note that AmiQuote puts a checkmark before ticker name in the list view. ONLY MARKED items will be downloaded. This allows you to perform selective downloads/updates.

**View→Refresh**

Basically AmiQuote handles refreshes by itself when needed. For example if you changed the date range, the list will be refreshed before starting download. But there are some cases when you may want to refresh the list by yourself. For example if you downloaded and imported quotes once and want to do this again you would need to choose this function. The Refresh function simply applies all date and type settings to the URLs listed, and MARKS all tickers for a new download.

**Tools→Import into AmiBroker**

This function is useful if you want to import just downloaded quotes into AmiBroker but you have **Automatic import** checkbox cleared.

**Tools→Get tickers from AmiBroker**

This function retrieves all symbols from currently loaded AmiBroker database and fills the AmiQuote ticker list with them.

**Tools→Get last update date**

This function retrieves the date of the most recent quotation of the first symbol present in currently loaded AmiBroker database and sets the **From** date to this date.

**Tools→Settings**

Displays the settings window where you can define the destination directory where all downloads are stored. Note that blank destination directory means that downloads will be stored in the current working directory (in most cases this is the folder from where current .TLS file was loaded).

In this window you can also change the mode of writing the files. By default historical files are overwritten while daily files get appended. This is recommended setup. Appending daily files simply allows you to create intra-day historical files when you do the updates daily. You may change this behaviour for your particular purpose.

## How to download quotes manually using AmiQuote

### Introduction

The purpose of this document is to explain how to use AmiQuote and AmiBroker in order to obtain quotes from Yahoo finance and Quote.com sites. AmiQuote is a companion program to AmiBroker charting/analysis software. The main purpose of AmiQuote is to simplify and automate downloading daily and historical quotation data from free Yahoo! Finance (USA, major European exchanges and some other countries), Quote.com (USA only) sites, MSN (USA and some European exchanges), Integratir (US stocks), Forex (Finam free site)

Yahoo provides data in "Historical" and "Current" modes of AmiQuote. Quote.com provides data in "Intraday" mode of AmiQuote.

### Preparing ticker list

A ticker list is a simple text file which lists line by line the tickers you want to import. The AmiQuote ticker list file has .TLS extension. AmiQuote comes with pre-written ticker list for components of main NYSE and NASDAQ indices and a number of European indices/markets. Additional ticker lists are available on the starter page at: <http://www.amibroker.com/starter/>. You can use those pre-written ticker lists or you can customize them or write your own one. In order to edit existing .TLS file or write completely new one all you need is plain text editor such as Notepad or any other plain ASCII editor (not MS Word!). All you have to do is to write tickers you want to import line by line (single ticker in single line) and save the file. Please make sure that you are saving the file with .TLS extension. Otherwise AmiQuote will not load this file.

Please note that Yahoo uses suffixes for non-US stocks. So in order to get quotes for non-US symbol you would need to add appropriate suffix to the ticker symbol. The suffixes in alphabetical order are (you can click on link to get the symbol list for each exchange) : [.AS – Amsterdam](#), [.AX – Australia \(ASX\)](#), [.BC – Barcelona](#), [.BE – Berlin](#), [.BO – Bombay](#), [.BM – Bremen](#), [.BR – Brussels](#), [.BA – Buenos Aires](#), [.CL – Calcuta](#), [.CR – Caracas](#), [.V – CDNX](#), [.CO – Copenhagen](#), [.D – Dusseldorf](#), [.F – Frankfurt](#), [.H – Hamburg](#), [.HA – Hanover](#), [.HK – Hong Kong](#), [.I – Ireland](#), [.JK – Jakarta](#), [.KA – Karachi](#), [.KQ – Kosdaq](#), [.KS – KSE](#), [.KL – Kuala Lumpur](#), [.L – London](#), [.LM – Lima](#), [.LS – Lisbon](#), [.MA – Madrid](#), [.MX – Mexico](#), [.MI – Milan](#), [.MU – Munich](#), [.NS – NSE](#), [.NZ – New Zeland](#), [.OL – Oslo](#), [.PA – Paris](#), [.SN – Santiago](#), [.SS – Shanghai](#), [.SZ Shenzhen](#), [.ST – Stockholm](#), [.SG – Stuttgart](#), [.TW – Taiwan](#), [.TA – Tel Aviv](#), [.TO – Toronto](#), [.VA – Valencia](#), [.VI – Viena](#), [.DE – XETRA](#), [.S – Zurich](#).

We will be soon offering Yahoo-compatible ticker lists for major European exchanges.

Please note that also Yahoo and Quote.com use different symbols for indices. The main difference is that Yahoo uses ^ (dash) prefix and Quote.com uses \$ (dollar) prefix.

For list of indices provided by **Yahoo** please [click here](#).

For list of indices provided by **Lycos/Quote.com** please [click here](#). Please note that recently Lycos/Quote.com stopped delivering free quotes and you need to have Livecharts subscription (\$9.95/month) in order to use it. For more details see [this Knowledge Base article](#).

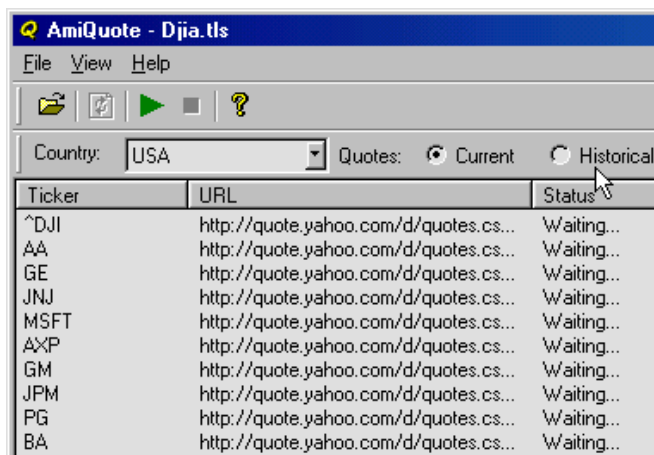
For list of symbols provided by **MSN** please [click here](#).

**Downloading data**

In order to download the data please launch AmiQuote. Then please click on "Open" button in the toolbar (or choose **File**→**Open** menu) as shown in picture on the right. ▶



From the file dialog please choose one .TLS file (for example DIJA.TLS) and click **Open** button. Then you will see the main screen of AmiQuote filled with the list of tickers loaded, as shown in picture below.



Choose appropriate Data Source

- **Yahoo Historical** – allows you to download end-of-day histories upto current day (current day data appear few hours after session end)
- **Yahoo Current** – allows you to download current day quotes (15-min delayed) during the trading session
- **Lycos/Quote.com Intraday** – allows you to download intraday and daily historical data (1-min bars and up) – for US stocks/futures only. If you have chosen this mode you should also select the bar interval (see the limitations described below) – need Livecharts subscription (\$9.95/month)
- **MSN Historical** – allows you to download end-of-day histories upto current day (current day data appear few hours after session end)
- **Forex** – allows you to download end-of-day and intraday (registered version) histories for the following currency pairs: EURCHF, EURGBP, EURJPY, EURUSD, GBPUSD, USDCHF, USDJPY

After choosing correct options please click on green arrow (or use **File** → **Start Download** menu). The download process will begin. AmiQuote will display progress messages and status information including number of completed downloads and number of files left. At anytime you can stop download process with "Stop" button (red box). After finishing the download **AmiQuote will automatically update the quotes in AmiBroker** (if only AmiBroker is running in parallel and "automatic import" box in AmiQuote is checked).

**Limitations**

Intraday interval bar data (1-min, 5-min, 15-min, 60-min and 120-min) are available for US securities only. Historical data for international exchanges are usually much shorter than for US markets.

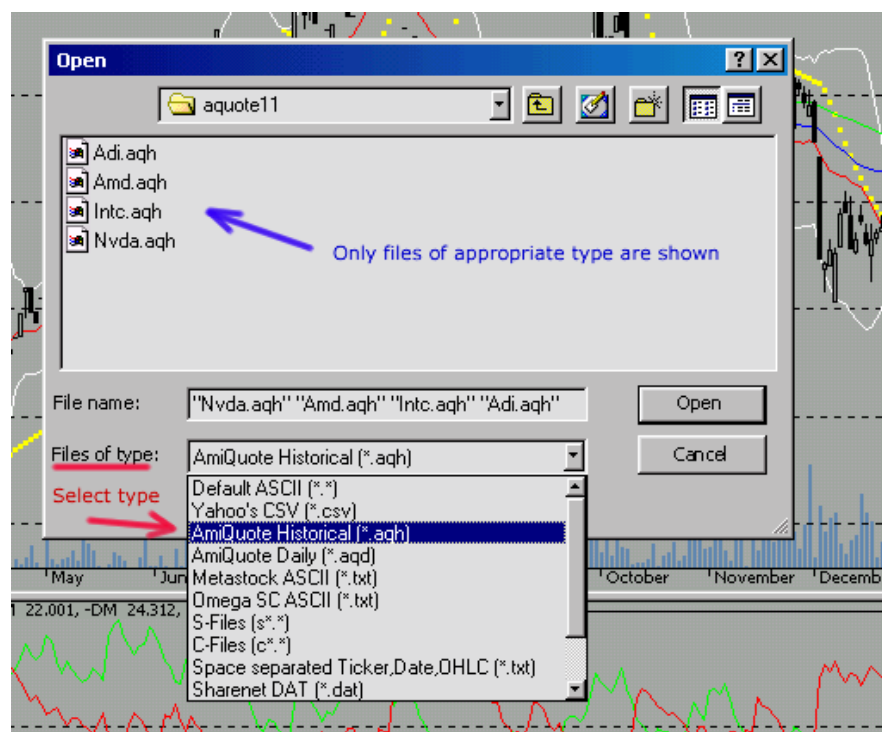
Because intraday bar data are downloaded from Quote.com servers the ticker symbols for indices are different than those used by Yahoo. For complete reference please check [http://finance.lycos.com/home/misc/symbol\\_search.asp?options=i](http://finance.lycos.com/home/misc/symbol_search.asp?options=i)

Intraday bar data are limited to 500 bars regardless of bar interval. In other words you always get 500 bars data, whenever these are 1-min, 5-min, 15-min, 60-min or 120-min data – so by choosing bigger interval you get data from more days. This is the limitation imposed by Livecharts server.

### Importing quotes into AmiBroker

**NOTE: This step is no longer necessary if you are using "automatic import" feature of AmiQuote. The explanations are provided only for users wanting to import selectively or re-import files downloaded in the past.**

First, please launch AmiBroker. From the **File** menu please select **Import From ASCII** option. You will see the following file dialog:



In this picture I marked the most important items for easy identification. Marked with red is type selector combo-box ("**Files of type**"). In order to import AmiQuote files (those with .AQH and .AQD extensions) you should choose **AmiQuote Historical** or **AmiQuote Daily**, or **AmiQuote Intraday (.AQI)** or **AmiQuote MSN (.AQM)** or **AmiQuote eSignalCentral (.AQE)** from the combo box (red arrow shows those options).

After choosing right type you will see only files of appropriate type in the file list (blue arrow shows that). Now you can select one or more files from the list. Multiple selection is possible by holding CTRL key depressed while selecting the items with a mouse (you can also press SHIFT for choosing a range of files with a single click). Now when you are done choosing the files you want to import just click "**Open**" button. The import process will start and you will see progress bar showing the AmiBroker is importing the data. After finishing the import AmiBroker will automatically refresh symbol list and you will see updated tickers and charts. If anything goes wrong with the import process AmiBroker writes a log file called "import.log" and located in AmiBroker's main directory. You can watch this log file if you want to find out what went wrong (since import.log is simple text file you can open it with any text editor)



## Common questions

Question	Answer
How can I edit my own ticker list (.TLS) file?	You can create or edit .TLS using Windows Notepad. When saving a file simply give .TLS extension to the file (instead of the default. TXT)
What about ready-to-use complete ticker lists for NYSE, NASDAQ, AMEX?	<p>There are following ready-to-use ticker lists available for download:</p> <ul style="list-style-type: none"> <li>• <a href="#">DJIA.TLS</a> (30 stocks)</li> <li>• <a href="#">DJTA.TLS</a> (20 stocks)</li> <li>• <a href="#">DJUA.TLS</a> (15 stocks)</li> <li>• <a href="#">NASDAQ100.TLS</a> (100 stocks)</li> <li>• <a href="#">NYSE.TLS</a> (2612 stocks)</li> <li>• <a href="#">NASDAQ.TLS</a> (4464 stocks)</li> <li>• <a href="#">AMEX.TLS</a> (794 stocks)</li> </ul>

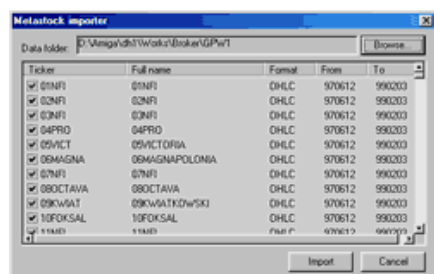
## Further information

For further information please consult AmiBroker User's Guide section "*Data management – Importing data from ASCII file*". In case of any further questions, comments and suggestions please contact me at: [support@amibroker.com](mailto:support@amibroker.com)

## Metastock importer window

**IMPORTANT NOTE:** Metastock importer should be used **ONLY** if you want to import MS data to native, local AmiBroker database once. If you want AmiBroker to just read Metastock database **DIRECTLY** without need to import new data over and over please set up your database **WITH METASTOCK PLUGIN** as described in the [Tutorial](#).

**NOTE 2:** if you setup your database with the **MS plugin** you should **NOT** use Metastock importer, because there is no point in using it when your data are already fed by the plugin.



Metastock importer opens AmiBroker to very rich source of historical data. The importer supports both old Metastock 6.5 and new 7.x (XMASTER) formats.

Basically Metastock data consist of:

- MASTER/EMASTER file which holds general information about the tickers, stock names, etc.
- F1.DAT....Fxx.DAT files which hold actual quotation data

The MASTER/EMASTER file is essential because it holds the references to Fxx.DAT files. Fxx.DAT files store only quotations in either 5 field (date/high/low/close/volume), 6 or 7 field (date/open/high/low/close/volume/openinterest) format. As you see MASTER/EMASTER and Fxx.DAT files are closely connected and you need them all to import the data.

## Usage

To import Metastock data you should do the following:

- Choose *Metastock import* from the menu
- Using the directory requester (**Browse...**) select the location of data in Metastock format (the directory with MASTER/EMASTER and Fxx.DAT files)
- After choosing proper directory AmiBroker will display the list of available symbols and date ranges. By default all available symbol will be marked for importing (checkmark at the beginning of the list). Now you can exclude some symbol from the import list by clicking appropriate item in the list (checkmark will toggle when you click).
- You can decide to which group and watch list the new symbols are added using **Group** and **Watch List** combos.
- After making your selections push '**Import**' button to start the process of importation.
- During the process you can cancel the operation by clicking '**Abort**' button in the progress window

## Understanding AmiBroker database concepts

### Background

A typical Windows application, for example, Paint, works with a SINGLE file. You just open and save that single file (.BMP in Paint, or .DOC in MS WORD), and that file holds all the necessary information.

AmiBroker is a more complex piece of software. It uses huge amounts of data (all quotes from different tickers, hand drawn studies, assignments to groups, markets, watch lists, favorites, industries, sectors, etc.), so it must manage multiple files.

It would actually be possible to save all this information in a single file, but it would be (a) huge, and (b) slow to update selectively. So AmiBroker uses multiple files for storing all the data. There are a lot of files associated with any database. The files for a particular database are stored in a directory (and its subdirectories) specific to that database. In AmiBroker documentation, such a directory is referred to either as a "database directory" (versions 3.9 or later) or as a "workspace directory" (earlier AmiBroker versions).

When you install AmiBroker for the first time, a default database directory is created, called 'data', in the AmiBroker directory. This database directory contains a sample Dow Jones Industrial Average database.

**In AmiBroker database menu and dialog selections, you are choosing or creating a database directory, not an individual file.**

### AmiBroker database structure

A database (or a workspace) is a directory that holds a set of binary files, which are stored in 0–9, a–z, '\_' subdirectories. Those binary files hold quotes, symbol information, your studies (trend lines, Fibonacci stuff). Each symbol's information is stored separately in the file with the name of the ticker symbol located in the subdirectory corresponding to the first character of the symbol, so IBM quotation data/studies are stored in the 'IBM' file located in the 'I' subdirectory.

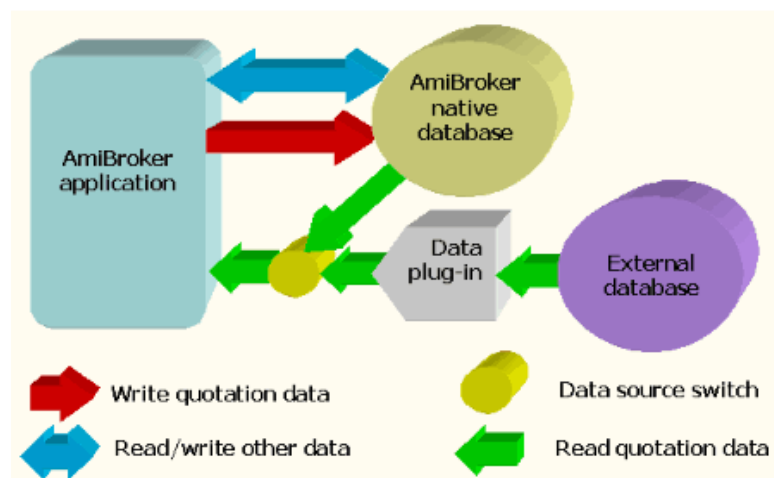
The default database for AB is the 'data' directory. It contains DJIA sample data. You may create additional databases in other directories via the File→'New database' menu.

In addition to these subdirectories and files, two additional files are also created by AmiBroker: broker.workspace and broker.master. The first is used to store category names and information about advancing/declining/unchanged issues. The latter stores the table of all symbols that is used for quick loading of the database. These two files are located in the root directory of each database, the 'data' directory, by default.

**In almost all cases, you should NOT touch files in an AmiBroker database, as the program manages them automatically, and no user intervention is required.**

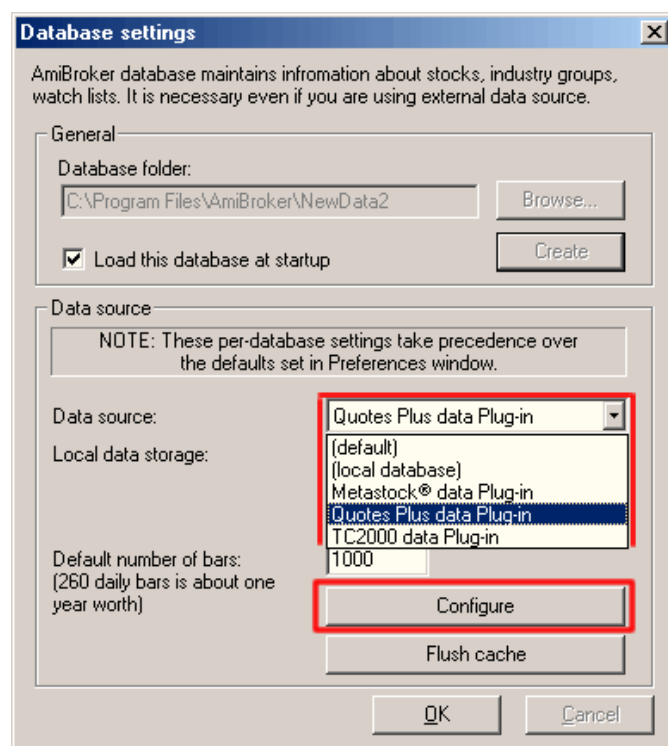
### What about the external data?

AmiBroker 3.9 has the ability to read quotes DIRECTLY from an external data source. Currently, AmiBroker can read directly from Quotes Plus (QP2), TC2000 (TC2K) and Metastock (MS) databases. This is achieved by means of data plug-ins that AmiBroker uses to read the data from an external data source. When a user decides that she/he wants to use an external database – AmiBroker – instead of reading the quotes from its own database – just asks the plug-in for quotes for any given symbol. The plug-in reads the external database and feeds the data to AmiBroker. The whole process is shown in the picture below:



As you can see, data plug-ins provide **read-only** access to the quotes in the external database. This means that your external data sources are never modified by AmiBroker. Changes or additions that you make to data and charts (like hand drawn studies, assignments to categories, etc.) are always saved in AmiBroker's own database. **So AmiBroker still uses its own database (to save changes, as a cache to speed up access, and for other tasks), even when using an external data source.**

The Data source switch represented in the graphic above can be set by the user to access various external databases. External data sources are selected by going to the File→'Database settings' dialog, shown below:



You may also choose to store the quotes retrieved from the external source to AmiBroker's own database for faster retrieval in subsequent accesses. If you want to do this, you should switch the 'Local data storage' setting to 'Enabled'.

Note: Similar settings can be found in the Tools->Preferences 'Data' tab, but these are only defaults used when creating new databases. **File->Database Settings configurations always take precedence over those done in Preferences -- EXCEPT in the following cases: If you choose the 'Default' entry in the Data Source drop down list (shown above), or the 'Default' radio button for Local Data Storage (also shown above), AmiBroker will use your Preferences settings for those items.**

## Understanding categories

AmiBroker has an ability to assign symbols to different categories allowing you (when properly set up) to narrow your analysis searches to the symbols meeting certain selection criteria (thanks to Filter feature available in Quick Review and Automatic Analysis windows). The initial set up of categories may be a little bit complicated especially when you want to track several thousands symbols.

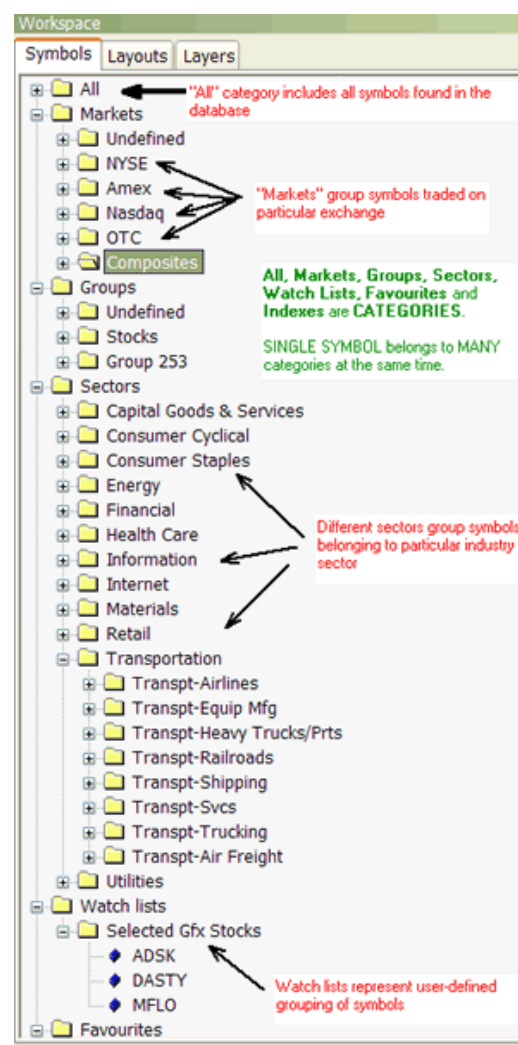
Categories show up in **Symbols** tab in **Workspace** window. First and foremost thing to remember is categories do NOT work like folders and Workspace window does NOT work like Windows Explorer.

The difference is fundamental. In the Windows Explorer file appears (usually) only once in the given tree leaf. In the workspace symbol tree given symbol shows up multiple times because it appears in every category leaf to which it belongs to even if this is the same symbol and only it exists only as single entity.

Single symbol belongs to MANY categories at the same time. For example AAPL (Apple Inc.) will belong to:

- *Stocks* group category
- *Nasdaq* market category
- *Information* sector category
- *Comp-Computer Mfg* industry category

and may also belong to several watch lists and favorites category. All at the same time. That's why one symbol will appear in many leaves of the workspace symbol tree. Now if you delete the SYMBOL it will of course disappear from ALL categories because you have deleted the symbol itself, not its assignment to category.



There are two types of categories:

1. with mutually exclusive membership: groups, markets, sectors/industries – it means that symbol must belong to single group, single market and single sector/industry at a time. You can move the symbol from one group/market/sector/industry to another but you can not remove this assignment – you should create "Unassigned" group/market/sector/industry instead and move 'unassigned' symbols there.
2. with free membership: watch lists/favorites/indexes– it means that a symbol may belong to ANY number (including zero) of watch lists (and to favorite/index category too). In this case you can remove this assignment by Watch List→Remove

Watch lists are covered in detail in the User's Guide: [Tutorial: Working with Watch Lists](#).

There is also one special category called "ALL" that shows up in the workspace symbol tree. It simply lists ALL symbols present in the database.

## Working with sectors and industries

### Basics – predefined sectors and industries

Now we will focus on setting up sectors and industries and assigning the symbols to them. First let me discuss some basic ideas.

AmiBroker comes with an example Dow Jones Industrials database holding all 30 components of this world's most famous market average. They are assigned to predefined sectors and industries. These sectors and industries are exactly the same as used on Yahoo finance site and here is a table which shows them all:

Sector	Industry
Basic Materials (0)	Chemical Manufacturing
	Chemicals – Plastics & Rubber
	Containers & Packaging
	Fabricated Plastic & Rubber
	Forestry & Wood Products
	Gold & Silver
	Iron & Steel
	Metal Mining
	Misc. Fabricated Products
	Non–Metallic Mining
	Paper & Paper Products
Capital Goods (1)	Aerospace & Defense
	Constr. – Supplies & Fixtures
	Constr. & Agric. Machinery
	Construction – Raw Materials
	Construction Services
	Misc. Capital Goods
	Mobile Homes & RVs
Conglomerates (2)	Conglomerates
Consumer Cyclical (3)	Apparel/Accessories
	Appliance & Tool
	Audio & Video Equipment
	Auto & Truck Manufacturers
	Auto & Truck Parts
	Footwear
	Furniture & Fixtures
	Jewelry & Silverware
	Photography
	Recreational Products
	Textiles – Non Apparel
	Tires

Consumer/Non-Cyclical (4)	Beverages (Alcoholic)
	Beverages (Non-Alcoholic)
	Crops
	Fish/Livestock
	Food Processing
	Office Supplies
	Personal & Household Prods.
	Tobacco
Energy (5)	Coal
	Oil & Gas – Integrated
	Oil & Gas Operations
	Oil Well Services & Equipment
Financial (6)	Consumer Financial Services
	Insurance (Accident & Health)
	Insurance (Life)
	Insurance (Miscellaneous)
	Insurance (Prop. & Casualty)
	Investment Services
	Misc. Financial Services
	Money Center Banks
	Regional Banks
	S&Ls/Savings Banks
Healthcare (7)	Biotechnology & Drugs
	Healthcare Facilities
	Major Drugs
	Medical Equipment & Supplies
Services (8)	Advertising
	Broadcasting & Cable TV
	Business Services
	Casinos & Gaming
	Communications Services
	Hotels & Motels
	Motion Pictures
	Personal Services
	Printing & Publishing
	Printing Services
	Real Estate Operations

	Recreational Activities
	Rental & Leasing
	Restaurants
	Retail (Apparel)
	Retail (Catalog & Mail Order)
	Retail (Department & Discount)
	Retail (Drugs)
	Retail (Grocery)
	Retail (Home Improvement)
	Retail (Specialty)
	Retail (Technology)
	Schools
	Security Systems & Services
	Waste Management Services
Technology (9)	Communications Equipment
	Computer Hardware
	Computer Networks
	Computer Peripherals
	Computer Services
	Computer Storage Devices
	Electronic Instruments & Controls
	Office Equipment
	Scientific & Technical Instr.
	Semiconductors
	Software & Programming
Transportation (10)	Air Courier
	Airline
	Misc. Transportation
	Railroads
	Trucking
	Water Transportation
Utilities (11)	Electric Utilities
	Natural Gas Utilities
	Water Utilities

It is important to understand the difference between a sector and an industry: industries "belong" to sectors, for example: "Air Courier", "Airline", "Railroads", "Trucking" industries belong to "Transportation" sector. So if a symbol is assigned to given industry, it is "automatically" assigned also to the corresponding sector.

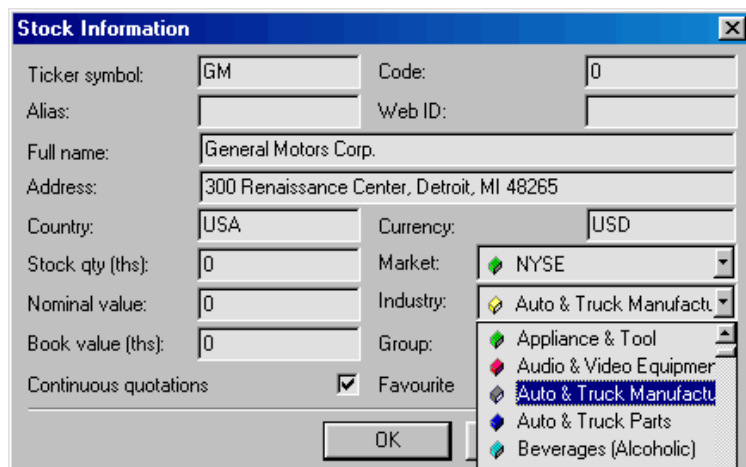


In the example DJIA database each stock is assigned to specific industry, for example GM (General Motors) is assigned to "Auto & Truck Manufacturers" industry, and this implicates that GM belongs to "Consumer/Cyclical" sector.

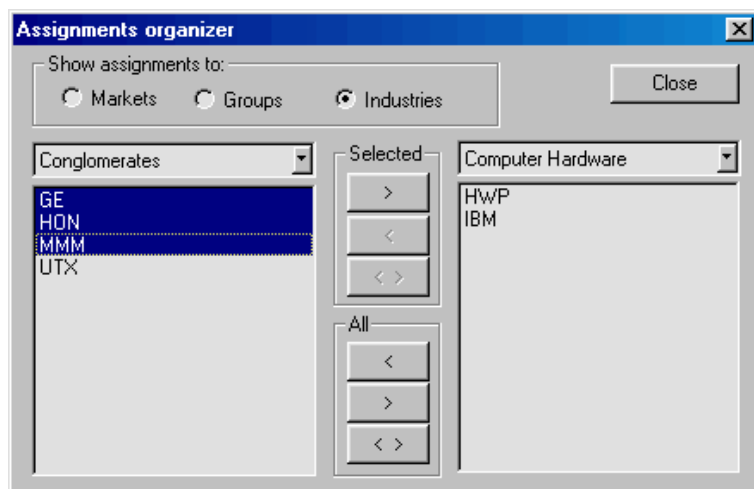
AmiBroker can handle up to 32 sectors and up to 256 industries.

### How to assign symbol to the industry?

You can change the industry to which given symbol is assigned by using **Symbol→Information** dialog (Industry combo box)



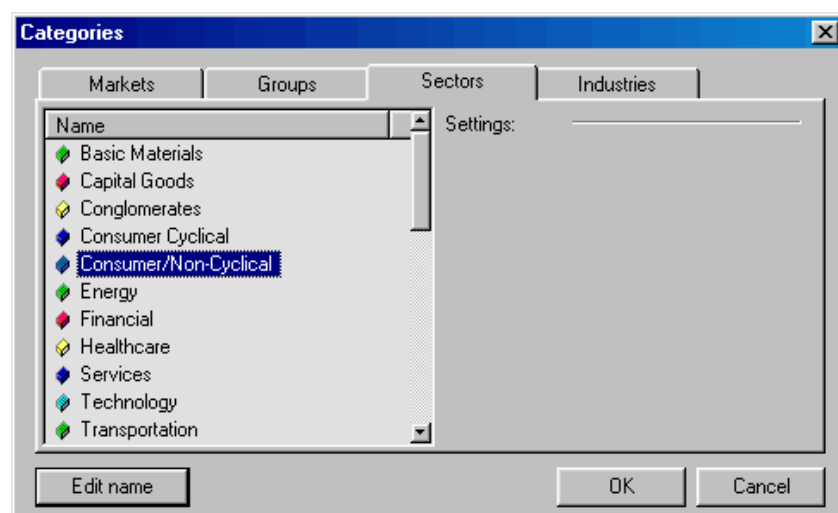
or using **Symbol→Organize Assignments**.



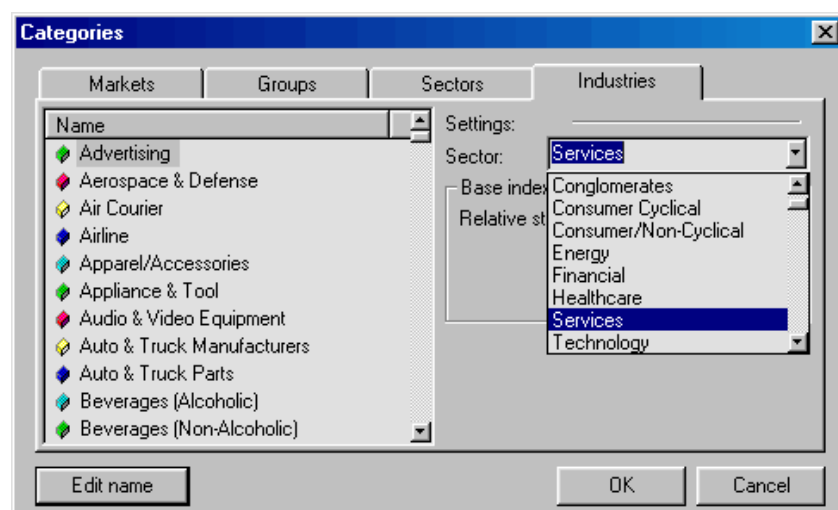
The first method is fine if you want to change single symbol settings. The latter is better if you want to move multiple symbols from one category to another.

### How to define your own sectors and industries

Please go to **Symbol→Categories** dialog, the last two tabs are "Sectors" and "Industries". First, switch to the "Sectors" tab and you will see the list of 32 sector names. You can now select the sector by clicking once on its name and edit the sector name by pressing ENTER or clicking "Edit name" button. Hit ENTER again to accept the name change.



After you renamed the sectors you can switch to the "Industries" tab. Similarly to the previous tab you can select the industry in the list and edit its name in the same manner. Here you can also assign the industry to the sector using "Sector" combo. Just select the sector to which you want to assign currently selected industry.



### Where sector and industry information is stored?

Generally speaking this information is stored in AmiBroker database. The sector and industry names and settings are stored in the broker.workspace file (in the workspace folder), symbol data files hold only the information about the assignment of the symbol to given industry (IndustryID).

When you create a new workspace (a database) AmiBroker sets up your industries and sectors according to the templates stored in the "broker.sectors" and "broker.industries" files. These are simple text files that could be edited with plain text editor (such as Notepad). These files could be also used for quick, automatic setup of the sectors and industries. AmiBroker comes with predefined broker.sectors and broker.industries that follow described above convention (see the table). You can rewrite broker.sectors and broker.industries files to define your own default scheme. So, "broker.sectors" and "broker.industries" files are used as a template when creating new workspace. Once workspace is created these files are **not** taken into consideration. In this way you may have different categories in each workspace. If you want AmiBroker to load them into already existing workspace please delete broker.workspace file **before** opening the workspace. If you then open the

workspace AmiBroker will read broker.sectors and broker.industries.

The layout of broker.sectors file is very simple: it is plain text file holding sector names written line by line as shown below:

```
Basic Materials
Capital Goods
Conglomerates
Consumer Cyclical
Consumer/Non-Cyclical
Energy
Financial
Healthcare
Services
Technology
Transportation
Utilities
```

The layout of broker.industries is similar, but in addition to industry names there is a number at the beginning of each line:

```
8 Advertising
1 Aerospace & Defense
10 Air Courier
10 Airline
3 Apparel/Accessories
3 Appliance & Tool
3 Audio & Video Equipment
3 Auto & Truck Manufacturers
3 Auto & Truck Parts
4 Beverages (Alcoholic)
4 Beverages (Non-Alcoholic)
7 Biotechnology & Drugs
8 Broadcasting & Cable TV
8 Business Services
8 Casinos & Gaming
0 Chemical Manufacturing
0 Chemicals - Plastics & Rubber
5 Coal
9 Communications Equipment
```

The numbers at front of industry names are "Sector IDs". Those numbers decide to which sector given industry belongs to. Because several industries may belong to one sector – you may need to put the same number for sector Id. Sector IDs are zero based, which means that 0 refers to the first line (sector name) of "broker.sectors" file, while 7 refers to the eighth line of this file. In the example above: "Advertising" industry belongs to "Services" sector, while "Aerospace & Defence" industry belongs to "Capital Goods" sector.

If you don't want to setup detailed industry information and want assign symbols only to sectors you can define one-to-one relationship between first 32 industries so they will be equivalent to sectors. Using the broker.sectors as show earlier in this article 1–1 broker.industries file would look like:

```

0 Basic Materials
1 Capital Goods
2 Conglomerates
3 Consumer Cyclical
4 Consumer/Non-Cyclical
5 Energy
6 Financial
7 Healthcare
8 Services
9 Technology
10 Transportation
11 Utilities

```

Note that this file is essentially the same as `broker.sectors` with the only difference that we have consecutive numbers prepended to each line. Using this kind of setup setting the industry will be equivalent to setting the sector.

### Making it automatic

As described above symbol and industries names and relationship can be easily set up quickly using "broker.sectors" and "broker.industries" files. It will save some work needed otherwise to enter this information in **Symbol->Categories** window.

Unfortunately a lot more work is needed to assign all symbols to the industries even using **Symbol->Organize Assignments** dialog. Fortunately there is a way to save a lot of work using AmiBroker automation interface and scripting. The detailed description of how to accomplish this task and ready-to-use script is presented in the [4th issue of AmiBroker Tips newsletter](#).

## Working with watch lists

AmiBroker 5.00 uses now new watch list system. Watch lists differ from other kinds of categories (as groups, markets, industries, sectors) in that, that you can assign single symbol to more than one watch list.

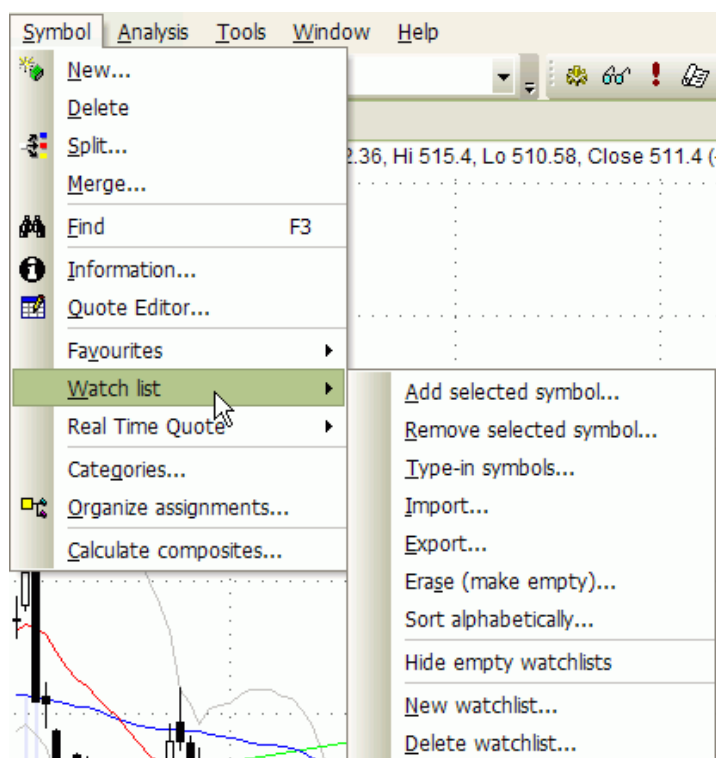
You can use UNLIMITED number of watch lists with their names definable in **Symbol->Categories** window. The members of each watch list is shown in the symbol tree under "Watch lists" leaf.

Watch lists are now stored as text files inside "Watchlists" folder inside database. The folder contains of any number of .TLS files with watch lists themselves and index.txt that defines the order of watch lists. You can add your own .tls file (one symbol per line) and AmiBroker will update index.txt automatically (adding any new watch lists at the end)The .TLS files can also be open in AmiQuote.

Watch lists remember the order in which symbols were added, so for example if you sort AA result list in some order and then you "add symbols to watch list" the order will be kept in the watch list.

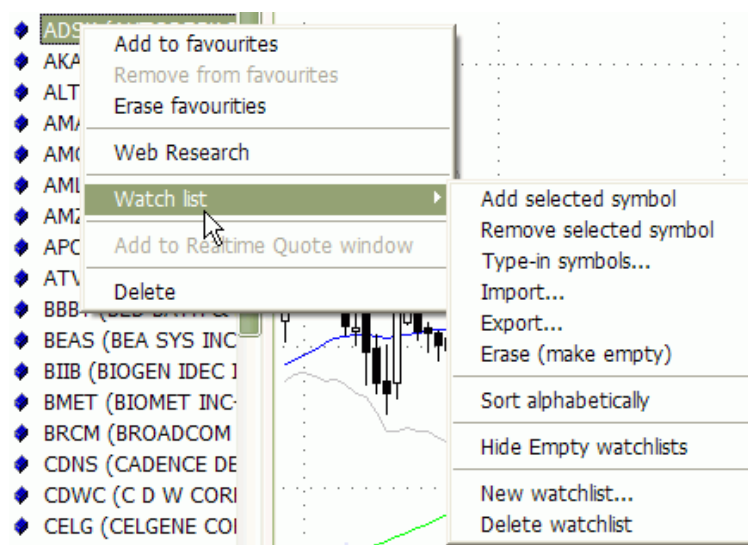
### *Adding / removing watch lists*

You can now Add/Delete watch lists using **Symbol->Watch List->New Watchlist** and **Symbol->Watch List->Delete Watch list** menu or from watch list context menu. Note that if you have done any customization to the menu, you may need to go to Tools->Customize, select "Menu Bar" and press "Reset" button for this new menu items to appear.

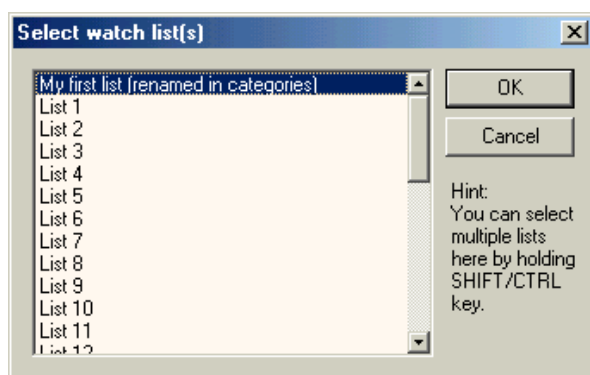


### Adding tickers to watch lists

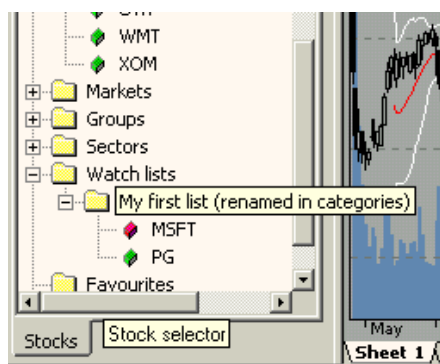
You can easily add a ticker to the watch list by simply clicking with a right mouse button over the item in the symbol tree and choosing **Watch List→Add selected symbol** option:



After choosing this option a watch list selector window will appear:



Here you should select the list you want to add the symbol to. Note that you can add one symbol to multiple lists at once, by holding CTRL key while clicking on the list items. After clicking OK selected symbol (MSFT) appears in the watch list of your choice:



You can also type-in symbols directly into the watch list using **Symbol->Watch list->Type-in option**. Symbols should be comma-separated. You can also right click over the watch list name in the workspace tree to type in symbols directly into the watch list.

### *Sorting tickers in a watch list*

You can now alphabetically sort the symbols in the watch list – click on the watch list with RIGHT mouse button and select **"Sort Alphabetically"**

### *Removing tickers from watch lists*

Removing symbols from the watch list is as easy as adding them. Just click on the list member with a right mouse button and select **Remove from watch list(s)**. Then similar list selector window will appear showing only those lists that currently selected symbol belongs to. You can now select one or more lists and the symbol will be removed from the list(s).

### *Erasing watch lists*

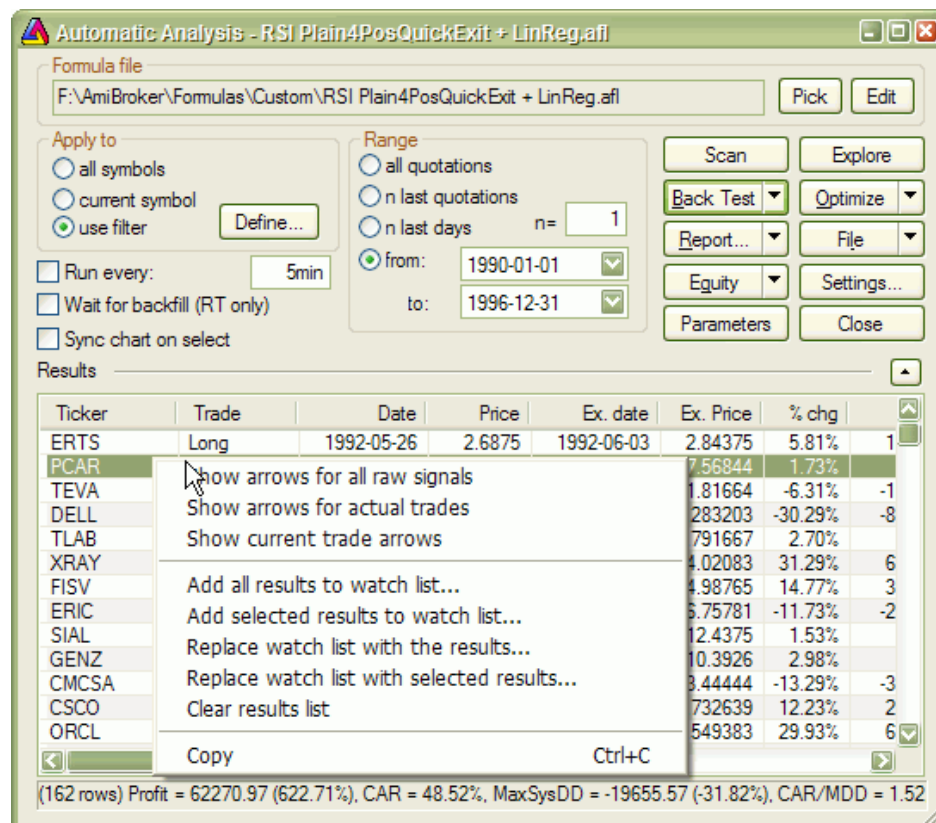
Sometimes you may want to clear (or erase) the whole watch list. Then just select **Symbol->Watch list->Erase (empty)** option. In the watch list selector window mark the list(s) you want to clear and click OK. This way selected watch list(s) become empty.

### *Hiding/Unhiding empty watch lists*

By default empty watch lists are shown in the symbol tree but you can hide them by right-click on watch list in the symbol tree and select **"Hide Empty Watchlists"** menu. To un-hide, select this option again.

### Using watch lists in Automatic analysis window

AmiBroker gives you a very easy way to store the results of scanning, backtesting and exploration into a watch list with a single mouse click – just run your favourite AFL formula over the whole database and click on the results list with a right mouse button to see the following menu:



When you choose **Add all/selected results to watch list** a watch list selector will appear where you select to which list symbols should be added, then after clicking OK all symbols filtered by your trading rules will automatically appear in the watch list of your choice.

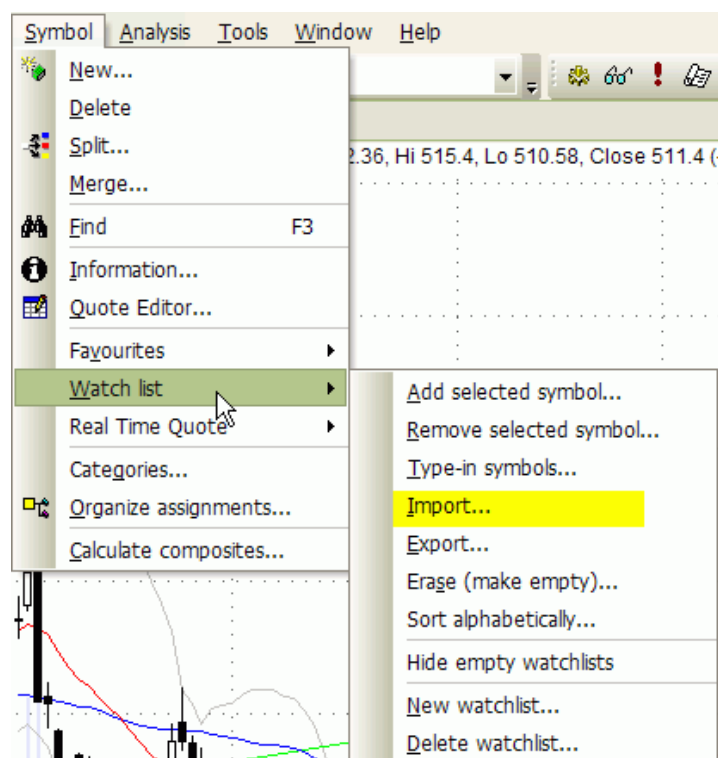
### You can also use option **Replace watch list with the results/selected results**

This new option empties the watch list before adding results. The order of symbols in the result list is preserved in the watch list.

## How to import/export watch list from/to file

### IMPORT WATCH LIST FROM FILE

1. Choose Symbol->Watch List->Import menu, or right click over watch list in the tree and choose Import.



2. Choose destination watch list

3. In the file dialog that will appear pick .TLS, .LST, .TXT or .CSV file

.TLS, .CSV, .TXT files should have **one ticker symbol per line and no other fields**.

.LST files are Quotes–Plus standard, comma separated list files that have the ticker symbol in the first column and some additional data in remaining columns. AmiBroker reads just first column and ignores rest.

4. Click OK.

### EXPORT WATCHLIST TO FILE

1. Choose Symbol–>Watch List–>Export menu.

or right click over watch list in the tree and choose Export.

2. Choose source watch list and switch to "External data source"

3. In the file dialog choose the file to export to. Generated file will be simple ASCII file with one ticker symbol per line.

---

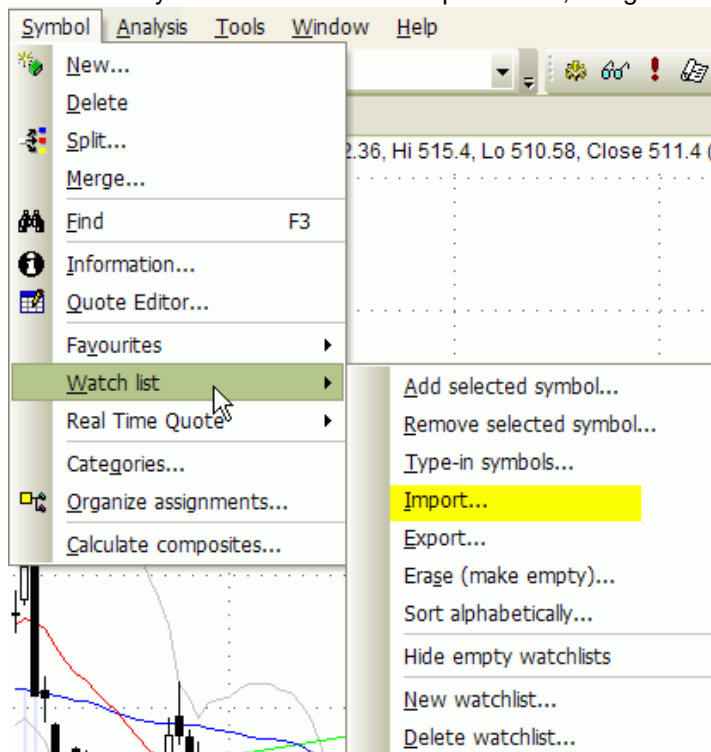
### How to import/export watch list from/to external database

**ATTENTION:** It works ONLY if you have "Data source" set to "Fast Track" plugin in File–>Database Settings (and if you have FastTrack database installed of course).

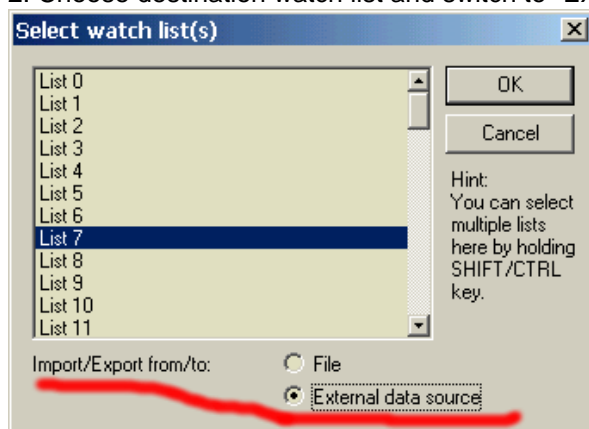
### IMPORT FAMILY FROM FASTTRACK



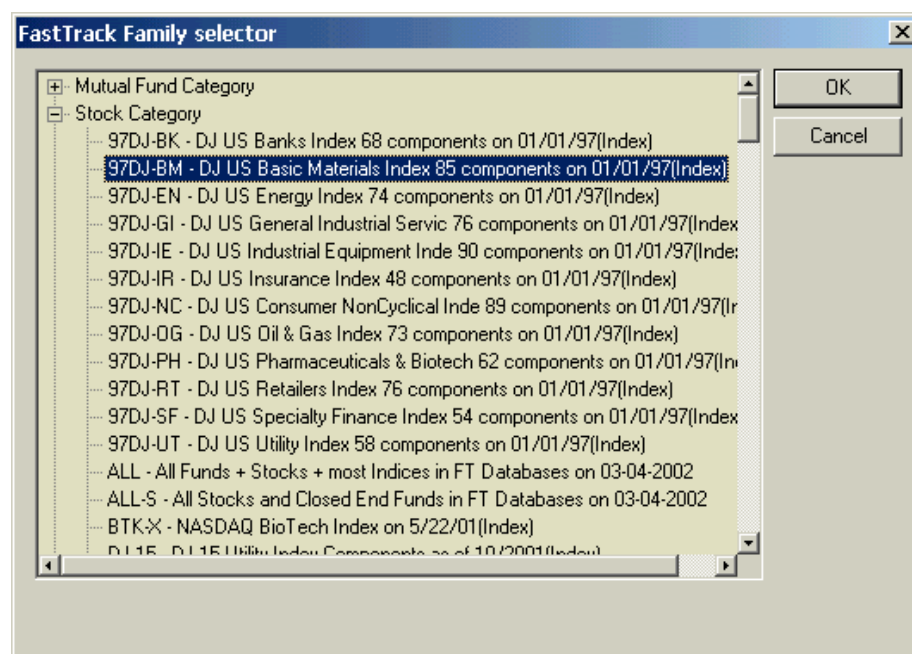
1. Choose Symbol->Watch List->Import menu, or right click over watch list in the tree and choose Import.



2. Choose destination watch list and switch to "External data source"



3. In the dialog that will appear unfold one category and select the family you want to import symbols from:

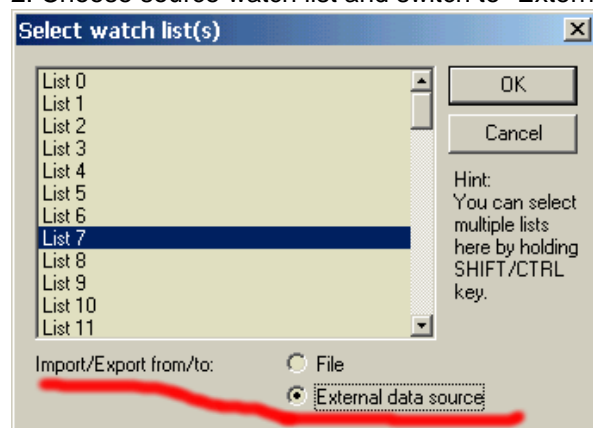


4. Click OK.

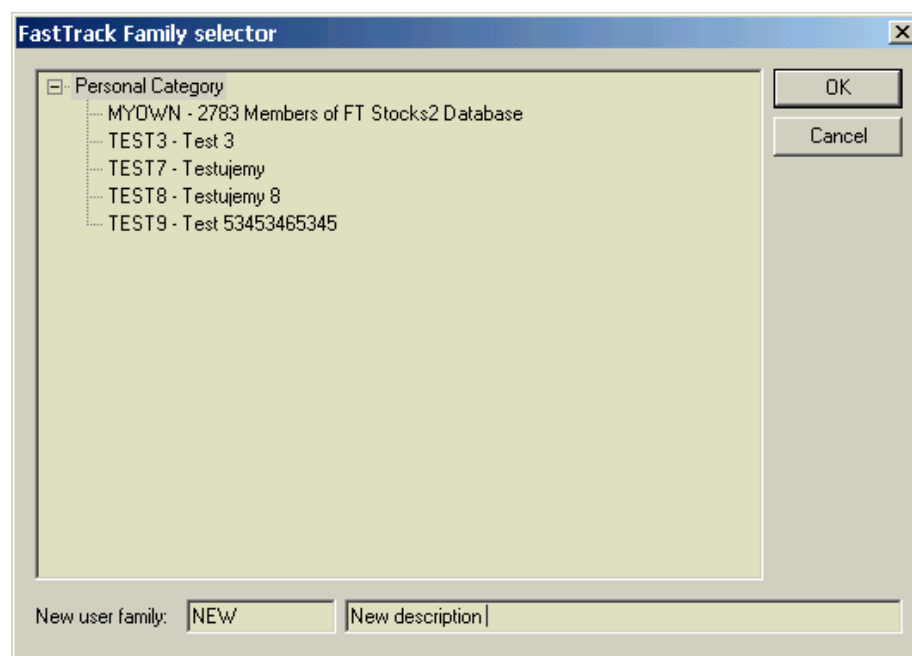
### EXPORT WATCHLIST TO FASTTRACK FAMILY

1. Choose Symbol→Watch List→Export menu.  
or right click over watch list in the tree and choose Export.

2. Choose source watch list and switch to "External data source"



3. Now either TYPE IN the new personal family name in "New user family" (and the description in the file next on the right side) OR choose existing personal family from the list.



## Understanding how AFL works

### Introduction

One of most important aspects of AFL is that it is an array processing language. It operates on arrays (or rows/vectors) of data. This way of operation is quite similar to the way how popular spreadsheets work (like Microsoft Excel). Anyone familiar with MS Excel should have no trouble quickly picking up AFL. – In fact all the examples in this article were all created using MS Excel.

### What is an Array?

An array is simply a list (or row) of values. In some books it may be referred to as a vector. Each numbered row of values in the example represents an individual array. Amibroker has stored in its database 6 arrays for each symbol. One for opening price, one for the low price, one for the high price, one for the closing price and one for volume (see the rows labelled 1–5 below) and one for open interest. These can be referenced in AFL as open, low, high, close, volume, openint or o, l, h, c, v, oi.

	Bar	1	2	3	4	5	6	7	8	9	10
1	Open	1,23	1,24	1,21	1,26	1,24	1,29	1,33	1,32	1,35	1,37

Fig 1. Open price array

Any other array is calculated from these 6 arrays using formulae built into AFL. These arrays are not stored in the database but calculated where necessary.

Each individual value in an array has a date associated with it. If you have the tool tip option turned on (Preferences → Miscellaneous Tab → Price data tool tips), when you move your cursor over candle on a

daily candle chart, a small yellow rectangle appears. AFL then looks up the open, low, high, close, volume values in the appropriate array and displays them inside the tool tip.

### **Processing arrays – why is AFL so fast?**

Lets see how the following statement is processed:

```
MyVariable = ( High + Low )/2;
```

When AFL is evaluating statement like this ( High + Low )/2 it does not need to re-interpret this code for each bar. Instead it takes the High ARRAY and Low ARRAY and adds corresponding array elements in single stage. In other words + operator (and other operators too) work on arrays at once and it is executed at full compiled-code speed, then the resulting array (each element of it) is divided by 2 also in single stage.

Let's look into the details – see fig 2.. When AFL engine looks at the ( High + Low )/2 it first takes High (1) and Low (2) arrays and produces (in single compiled step) the temporary array (3). Then it creates the final array (4) by dividing each element of temporary array by two. This result is assigned to myVariable

	Bar	1	2	3	4	5	6	7	8	9	10
1	<b>High</b> (built-in array)	1,24	1,27	1,25	1,29	1,25	1,29	1,35	1,35	1,37	1,29
2	<b>Low</b> (built-in array)	1,20	1,21	1,19	1,20	1,21	1,24	1,30	1,28	1,31	1,27
3	<b>High+Low</b> (temporary array created during evaluation)	2,44	2,48	2,44	2,49	2,46	2,53	2,65	2,63	2,68	2,46
4	<b>( High+Low ) /2</b> (gets assigned to MyVariable)	1,22	1,24	1,22	1,245	1,23	1,265	1,325	1,315	1,34	1,23

Fig 2. AFL steps when processing ( High + Low ) /2

### **Moving averages, conditional statements**

Let us now consider the following code:

```
Cond1 = Close > MA( Close, 3 );
Cond2 = Volume > Ref( Volume, -1 );
Buy = Cond1 AND Cond2;
Sell = High > 1.30;
```

This code generates a buy signal when todays close is higher than 3 day moving average of close AND todays volume is higher than yesterday's volume. It also generates a sell signal when today's high is higher than 1.30.

If in your AFL code you need to see if the closing price is greater than say a 3 day simple moving average AFL will first run through the close array creating a new array called MA(close,3) for the symbol being analysed. Each cell in the new array can then be compared one for one in the close array. In the example an array called Cond1 is created this way. For each cell where the closing price is greater than the corresponding cell value in MA(close,3) the cell value for new array 'Cond1' is set to '1'. If the closing price is not greater than the corresponding price in the close array the value in 'Cond1' is set to '0'.

AFL can also look forwards or backwards a number of cells in an array using the **Ref** function (see row 6 where temporary array is created holding previous day volume)

In row 9 a new array called Cond2 has been created by comparing the value of each cell in the volume array with its previous cell setting the Cond2 cell value to '1' if true and '0' if false.

Row 10 shows an array called 'Buy' created by comparing the cell values in Cond1 with the cell values in Cond2. If the cell in Cond1 has a '1' AND so does the corresponding cell in Cond2 then a '1' is placed in the 'Buy' array cell.

Row 11 shows an array called 'Sell' created whenever the cell value in the close array is greater than \$1.30.

	Day	1	2	3	4	5	6	7	8	9	10
1	Open	1,23	1,24	1,21	1,26	1,24	1,29	1,33	1,32	1,35	1,37
2	High	1,24	1,27	1,25	1,29	1,25	1,29	1,35	1,35	1,37	1,29
3	Low	1,20	1,21	1,19	1,20	1,21	1,24	1,30	1,28	1,31	1,27
4	Close	1,23	1,26	1,24	1,28	1,25	1,25	1,31	1,30	1,32	1,28
5	Volume	8310	3021	5325	2834	1432	5666	7847	555	6749	3456
6	Ref( Volume, -1 ) (temporary array created during eval)	Null	8310	3021	5325	2834	1432	5666	7847	555	6749
7	MA( Close, 3 ) (temporary array created during eval)	Null	Null	1,243	1,260	1,257	1,260	1,270	1,287	1,310	1,300
8	Cond1 = Close < MA(close,3) (gives 1 (or true) if condition met, zero otherwise)	Null	Null	1	0	1	1	0	0	0	1
9	Cond2 = Volume > Ref(volume,-1)	Null	0	1	0	0	1	1	0	1	0
10	Buy = Cond1 AND Cond2	Null	Null	1	0	0	1	0	0	0	0
11	Sell = High > 1.30	0	0	0	0	0	0	1	1	1	0

Obviously Buy and Sell are special arrays whose results can be displayed in the Analyser window or on screen using a red or green value as needed.

### Getting little bit more complex

The examples above were very simple. Now I will just explain 3 things that seem to generate some confusion among the users:

- referencing selected values (SelectedValue, BeginValue, EndValue, LastValue)
- IIF function
- AMA function

As written in the [Tutorial: Basic charting guide](#) you can select any quote from the chart and you can mark From-To range. The bar selected by vertical line is called "selected" bar while start and end bars of the range are called "begin" and "end" bars. AFL has special functions that allow to reference value of the array at selected, begin and end bar respectively. These functions are called SelectedValue, BeginValue and EndValue. There is one more function called LastValue that allows to get the value of the array at the very last bar. These four functions take the array element at given bar and return SINGLE NUMBER representing the value of the array at given point. This allows to calculate some statistics regarding selected points. For example:

```
EndValue( Close ) - BeginValue( Close )
```

Will give you dollar change between close prices in selected from-to range.

When number retrieved by any of these functions is compared to an array or any other arithmetic operation involving number and the array is performed it works like the number spanned all array elements. This is illustrated in the table below (rows 2, 6, 7). Green color marks "begin" bar and red color marks "end" bar. Selected bar is marked with blue.

	Day	1	2	3	4	5	6	7	8	9	10
1	Open	1,23	1,24	1,21	1,26	1,24	1,29	1,33	1,32	1,35	1,37
2	BeginValue( Open )	1,24	1,24	1,24	1,24	1,24	1,24	1,24	1,24	1,24	1,24
3	EndValue( Open )	1,32	1,32	1,32	1,32	1,32	1,32	1,32	1,32	1,32	1,32
4	SelectedValue( Open )	1,21	1,21	1,21	1,21	1,21	1,21	1,21	1,21	1,21	1,21
5	LastValue( Open )	1,37	1,37	1,37	1,37	1,37	1,37	1,37	1,37	1,37	1,37
6	Close	1,22	1,26	1,23	1,28	1,25	1,25	1,31	1,30	1,32	1,28
7	Close <= BeginValue( Open )	1	0	1	0	0	0	0	0	0	0
8	result = IIF( Close <= BeginValue( Open ), Close, Open );	1,22	1,24	1,23	1,26	1,24	1,29	1,33	1,32	1,35	1,37
9	Period	2	3	4	2	3	5	2	3	4	2
10	Factor = 2/(Period+1)	0,667	0,500	0,400	0,667	0,500	0,333	0,667	0,500	0,400	0,667
11	1 – Factor	0,333	0,500	0,600	0,333	0,500	0,667	0,333	0,500	0,600	0,333
12	AMA( Close, Factor )	0,8125	1,0363	1,1138	1,2234	1,2367	1,2399	1,2853	1,2927	1,3036	1,2866

Now the **IIF(condition, truepart, falsepart)** function. It works that it returns the value of second (*truepart*) or third (*falsepart*) argument depending on *condition*. As you can see in the table above in row 8 the values come from Close array (*truepart*) for bars when condition is true (1) and come from Open array (*falsepart*) for the remaining bars. In that case the array returned by IIF function consists of some values from Close and some values from Open array. Note that both *truepart* and *falsepart* are arrays and they are evaluated regardless of the condition (so this is not a regular IF-THEN-ELSE statement but **function** that returns array)

The **AMA( array, factor)** function seems to cause the most problems with understanding it. But in fact it is very simple. It works in recursive way. It means that it uses its previous value for the calculation of current value. It processes array bar by bar, with each step it multiplies given cell of first argument (array) by given cell of second argument (factor) and adds it to the previous value of AMA multiplied by (1-factor). Lets consider column 3. The value of AMA in the column 3 is given by multiplying close price from column 3 (1,23) by factor (0,4). Then we add the previous value of AMA (1,0363) multiplied by (1-factor = 0,6). The result (rounded to 4 places) is  $1,23 * 0,4 + 1,0363 * 0,6 = 1,1138$ .

If you look at the figures in the row 12 you may notice that these values look like a moving average of close. And that's true. We actually presented how to calculate variable-period exponential moving average using AMA function.

### New looping

With version 4.40 AmiBroker brings ability to iterate through quotes using *for* and *while* loops and adds *if-else* flow control statement. These enhancements make it possible to work BOTH ways: either use ARRAY processing (described above) for speed and simplicity or use LOOPS for doing complex things. As an

example how to implement variable period exponential averaging (described above) using looping see the following code:

```
Period = ... some calculation

vaexp[ 0 ] = Close[ 0 ]; // initialize first value

for( i = 1; i < BarCount; i++ )
{
    // calculate the value of smoothing factor
    Factor = 2/(Period[ i ] + 1 );

    // calculate the value of i-th element of array
    // using this bar close ( close[ i ] ) and previous average value ( vaexp[ i - 1 ] )
    vaexp[ i ] = Factor * Close[ i ] + ( 1 - Factor ) * vaexp[ i - 1 ];
}
```

As you can see the code is longer but on the other hand it is very similar to any other programming language as C/Pascal/Basic. So people with some experience with programming may find it easier to grasp.

If you are beginner I suggest to learn array processing first before digging into more complex looping stuff.

If you're having trouble coding AFL I suggest you generate the arrays in the example in Excel for yourself. If that's a problem get some help from a friend – especially if that friend is an accountant.

Once you've got the hang of it you can code any system from a book on trading – or build one yourself.

--- Special thanks to Geoff Mulhall for [original article in the newsletter](#) that was the basis of this tutorial ---

## Creating your own indicators

There are two ways to create your own indicators:

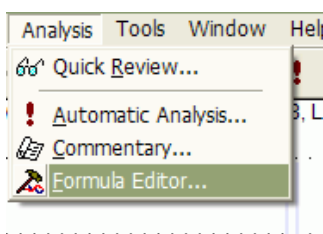
- 1) using drag-and-drop interface
- 2) by writing your own formula

First method, using drag-and-drop interface is very simple and does not require writing single line of code. To learn more about drag-and-drop indicator creation please check [Tutorial: How to use drag-and-drop charting interface](#)

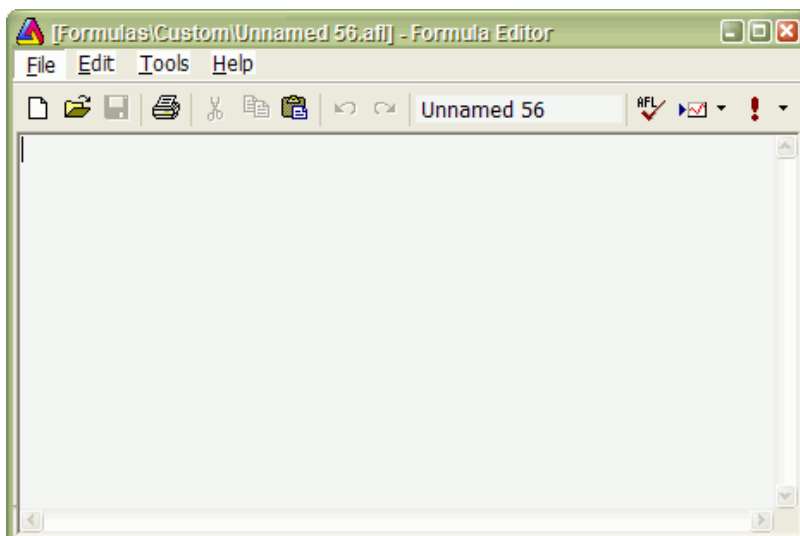
Second method involves writing an indicator formula in flexible AFL (AmiBroker Formula Language). You can find the description of this language in [AFL Reference Guide section of user's guide](#). Here we will present basic steps needed to define and display your own custom indicator. In this example we will define an "indicator" that will show line volume graph (opposite to built-in bar volume graph).

Just follow these steps

1. Select *Analysis->Formula Editor* option from the menu as shown below:

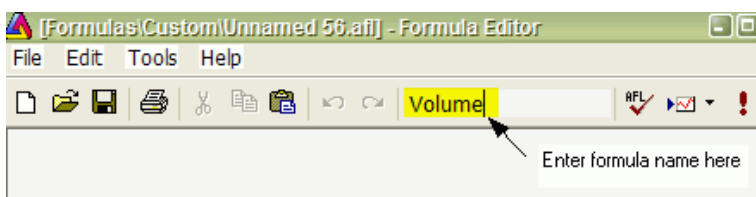


2. You will see the following dialog displayed on the screen:

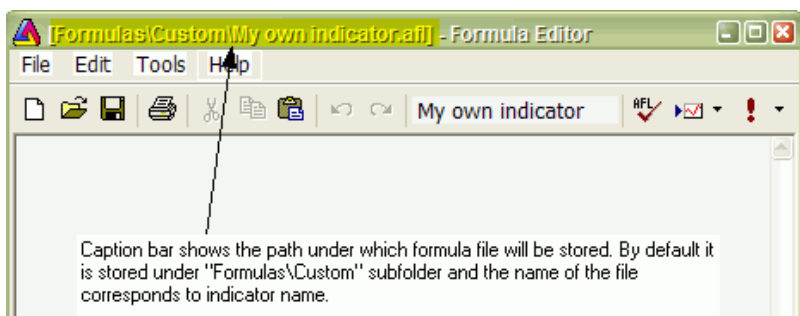


It presents an empty [Formula Editor](#) window.

3. Now single-click in the edit field located in the editor toolbar to change the name of the indicator:



Now you can edit the name of the custom indicator. Give it the name "My own indicator". After you press ENTER key the caption will be updated with the new file name as shown below:

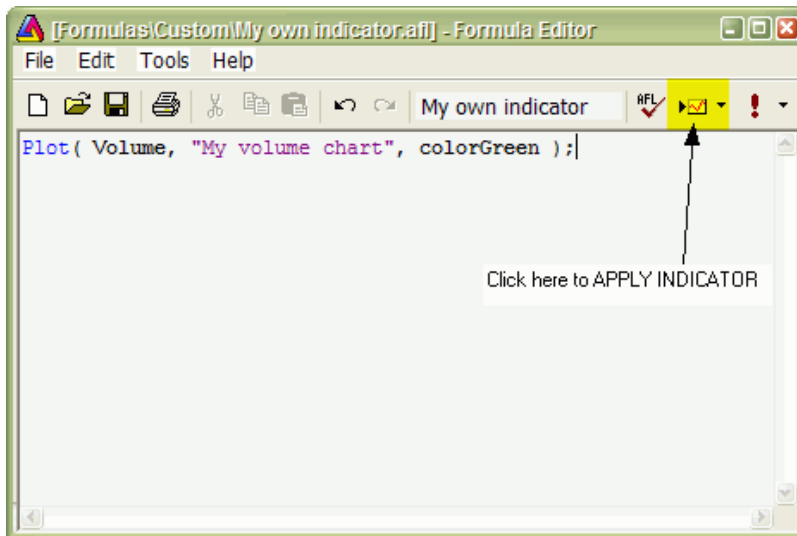


4. Now type the formula:



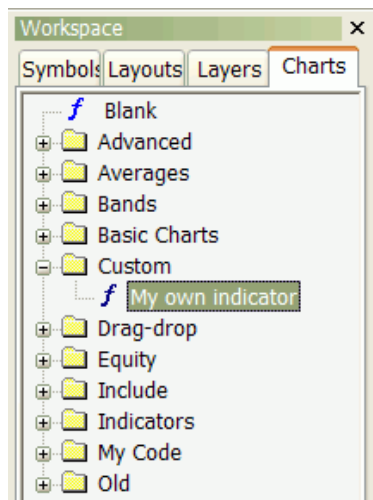
```
Plot( Volume, "My volume chart", colorGreen );
```

This formula instructs AmiBroker to plot built-in Volume array. Second parameter specifies the title of the plot and third parameter defines the color. The picture below shows formula editor after entering the code:

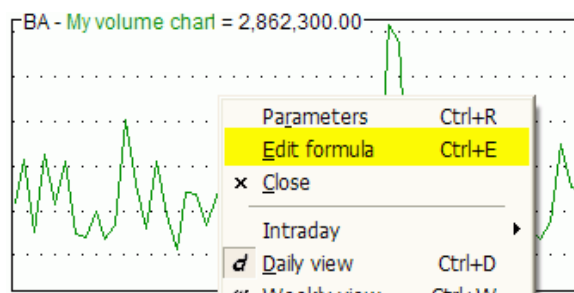


5. Now click **Apply indicator** toolbar button (or choose **Tools→Apply indicator** menu) as shown in the picture and close editor by pressing **X** button in the upper right corner of the editor window.

Now the indicator you have just written is displayed as a chart. You can also find it stored as a formula in Chart tree:



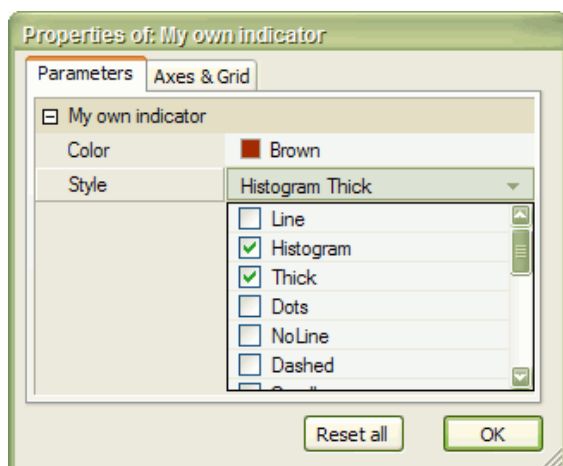
Now you can improve your indicator by adding Param functions so both color and style of the plot can be modified using **Parameters** dialog. To do so, click with **RIGHT** mouse button over chart pane and select **Edit Formula** (or press Ctrl+E)



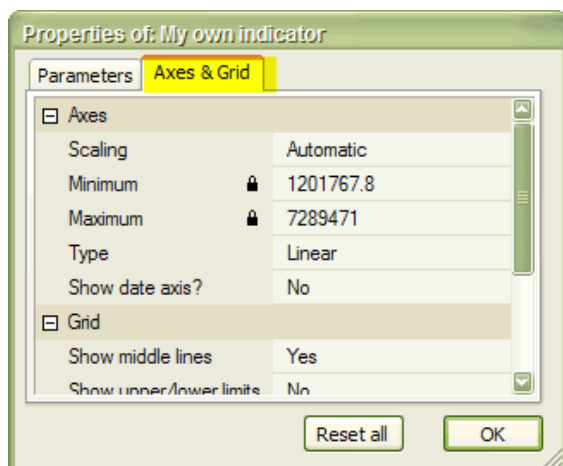
And modify the formula to:

```
Plot( Volume, "My volume chart", ParamColor("Color", colorGreen ),
ParamStyle("Style", 0, maskAll ) );
```

Then press **Apply indicator** to apply the changes. Now click with RIGHT mouse button over chart pane again and select **Parameters** (or press Ctrl+R) and you will see parameters dialog allowing to modify colors and styles used to plot a chart:



Also in the "Axes & Grid" tab you will be able to change settings for axes, grids and other charting options referring to this particular chart:



For further information on creating your indicators please check [Using graph styles and colors](#) tutorial section

For further reference on using Formula Editor please consult *Environment – Formula Editor* and *AmiBroker Formula Language – AFL Tools* sections of AmiBroker User's guide and using [AFL editor](#).

## Using graph styles, colors, titles and parameters in Indicators

AmiBroker provides customizable styles and colors of graphs in custom indicators. These features allow more flexibility in designing your indicators. This article will explain how to use styles and colors. It will also explain how to define chart title that appears at the top of the chart.

### Plot() function

Plot is the function used to plot a chart. It takes 6 parameters, out of which first 3 are required.

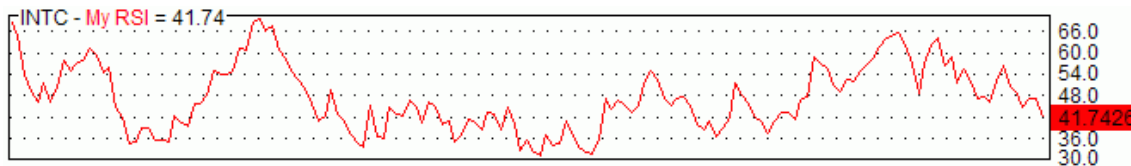
Plot( array, name, color, style = styleLine, minvalue = Null, maxvalue = Null, XShift = 0 )

- *array* parameter represents data to be plotted,
- *name* parameter defines the name of the graph (used in title string to show the values of the indicator),
- *color* parameter defines the color of plot,
- *style* defines "the look" of the chart (i.e. line/histogram/candlestick/bar, etc). Default style is line.
- *minvalue* and *maxvalue* are rarely used parameters that define hard-coded minimum and maximum values used when graph uses "independent" scaling, i.e. styleOwnScale is specified in *style* parameter. Usually you don't need to specify them at all.
- *XShift* allows shifting chart past the last bar (for example displaced moving averages or projections into the future)

An example, the following single function call plots a RSI indicator with red color line:

```
Plot( RSI(14), "My RSI", colorRed );
```

As you can see we have provided only first three (required) parameters. First parameter is the array we need to plot. In our example it is RSI(14) indicator. Second parameter is just the name. It can be any name you want. It will be displayed in the title line along with indicator value as shown in the picture below:



Third parameter is the color. To specify plot color you can use one of the following pre-defined constants:

#### Color constants

Custom colors refer to color user-defined palette editable using Tools->Preferences->Colors, the numerical values that appear after = (equation) mark are for reference only and you don't need to use them. Use just the name such as colorDarkGreen.

```
colorCustom1 = 0
colorCustom2 = 1
colorCustom3 = 2
colorCustom4 = 3
```

```
colorCustom5 = 4
colorCustom6 = 5
colorCustom7 = 6
colorCustom8 = 7
colorCustom9 = 8
colorCustom10 = 9
colorCustom11 = 10
colorCustom12 = 11
colorCustom13 = 12
colorCustom14 = 13
colorCustom15 = 14
colorCustom16 = 15

colorBlack = 16
colorBrown = 17
colorDarkOliveGreen = 18
colorDarkGreen = 19
colorDarkTeal = 20
colorDarkBlue = 21
colorIndigo = 22
colorDarkGrey = 23

colorDarkRed = 24
colorOrange = 25
colorDarkYellow = 26
colorGreen = 27
colorTeal = 28
colorBlue = 29
colorBlueGrey = 30
colorGrey40 = 31

colorRed = 32
colorLightOrange = 33
colorLime = 34
colorSeaGreen = 35
colorAqua = 35
colorLightBlue = 37
colorViolet = 38
colorGrey50 = 39

colorPink = 40
colorGold = 41
colorYellow = 42
colorBrightGreen = 43
colorTurquoise = 44
colorSkyblue = 45
colorPlum = 46
colorLightGrey = 47

colorRose = 48
colorTan = 49
```

```
colorLightYellow = 50
colorPaleGreen = 51
colorPaleTurquoise = 52
colorPaleBlue = 53
colorLavender = 54
colorWhite = 55
```

You can also use new 24-bit (full color palette) functions [ColorRGB](#) and [ColorHSB](#)

You can easily plot multi colored charts using both Plot functions. All you need to do is to define array of color indexes.

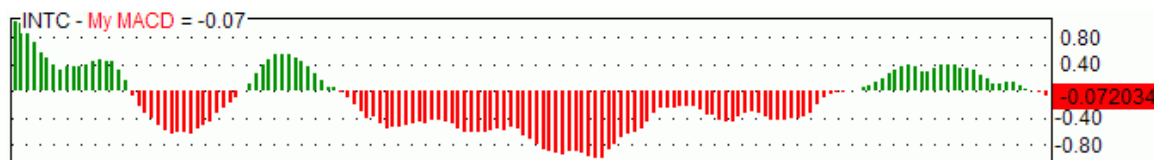
In the following example MACD is plotted with green color when it is above zero and with red color when it is below zero.

```
dynamic_color = IIf( MACD() > 0, colorGreen, colorRed );
Plot( MACD(), "My MACD", dynamic_color );
```

In addition to defining the color we can supply 4th parameter that defines style of plot. For example we can change previous MACD plot to thick histogram instead of line:

```
dynamic_color = IIf( MACD() > 0, colorGreen, colorRed );
Plot( MACD(), "My MACD", dynamic_color, styleHistogram |
styleThick );
```

As you can see, multiple styles can be combined together using | (binary-or) operator. (Note: the | character can be typed by pressing backslash key '\' while holding down SHIFT key). Resulting chart looks like this:



To plot candlestick chart we are using styleCandle constant, as in this example:

```
Plot( Close, "Price", colorBlack, styleCandle );
```

To plot traditional bars with color (green up bars and red down bars) we just specify color depending on relationship between open and close price and styleBar in style argument:

```
Plot( Close, "Price", IIf( Close > Open, colorGreen, colorRed ),
styleBar | styleThick );
```

All available style constants are summarized in the table below.

### Style constants

Style is defined as a combination (using either addition (+) or binary-or (|) operator) of one or more following flags ( you can use predefined style\_\_ constants instead of numbers)

styleLine = 1 – normal (line) chart (default)  
styleHistogram = 2 – histogram chart

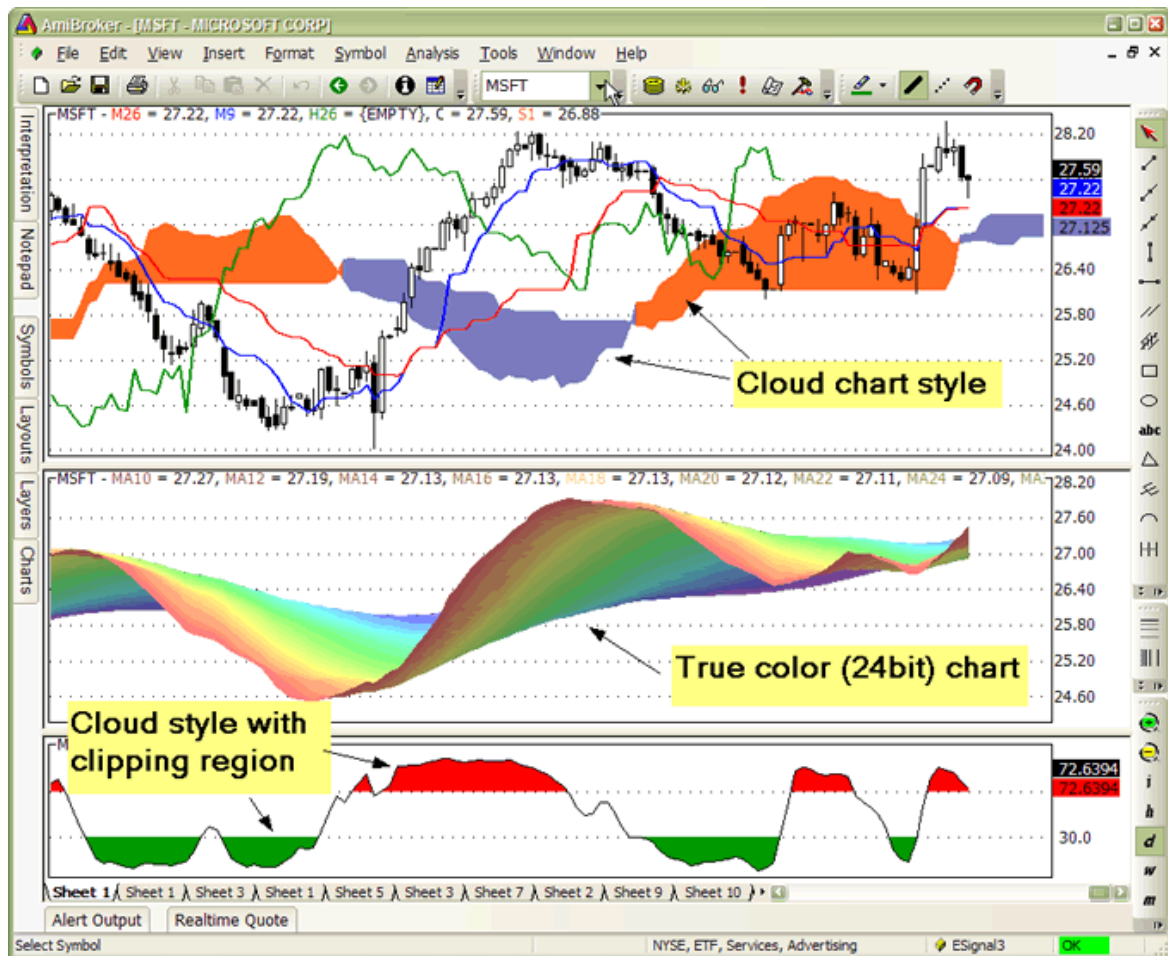
styleThick = 4 – fat (thick)  
styleDots = 8 – include dots  
styleNoLine = 16 – no line  
styleDashed = 32 – dashed line style  
styleCandle = 64 – candlestick chart  
styleBar = 128 – traditional bar chart  
styleNoDraw = 256 – no draw (perform axis scaling only)  
styleStaircase = 512 – staircase (square) chart  
styleSwingDots = 1024 – middle dots for staircase chart  
styleNoRescale = 2048 – no rescale  
styleNoLabel = 4096 – no value label  
stylePointAndFigure = 8192 – point and figure  
styleArea = 16384 – area chart (extra wide histogram)  
styleOwnScale = 32768 – plot is using independent scaling  
styleLeftAxisScale = 65536 – plot is using left axis scale (independent from right axis)  
styleNoTitle = 131072 – do not include this plot value in title string  
styleCloud = 262144 – paint a "cloud" (filled area) chart (see examples below)  
styleClipMinMax = 524288 – clip area between Min and Max levels defined in Plot statement. (Note: this style is not compatible with most printers)

Not all flag combinations make sense, for example (64+1) (candlestick + line) will result in candlestick chart (style=64)

Note on candlestick/bar charts: if these styles are applied to Plot() function then they use indirectly O, H, L arrays.

If you want to specify your own OHL values you need to use [PlotOHLC\(\)](#) function.

New styleCloud and styleClipMinMax styles bring new interesting possibilities shown in the sample image below:



The formula for chart in the middle pane (rainbow 24-bit multiple moving averages) looks as follows:

```
side = 1;
increment = Param("Increment", 2, 1, 10, 1);
for( i = 10; i < 80; i = i + increment )
{
    up = MA( C, i );
    down = MA( C, i + increment );

    if( ParamToggle("3D effect?", "No|Yes", 1) )
        side = IIf(up <= down AND Ref( up <= down, 1 ), 1, 0.6 );

    PlotOHLC( up, up, down, down, "MA"+i, ColorHSB( 3*(i - 10),
        Param("Saturation", 128, 0, 255 ),
        side * Param("Brightness", 255, 0, 255 ) ), styleCloud | styleNoLabel
    );
}
```

The formula for the chart in the lower pane (slow stochastic %K with colored tops and bottoms) looks as follows. It uses styleClipMinMax to achieve clipping of the cloud region between min and max levels specified in the plot statement. Without this style area between min/max would be filled. Please note that due to Windows GDI limitation clipping region (styleClipMinMax) is supported only on raster (bitmap) devices so it is not compatible with printers or WMF (windows metafile) output.

```
SetChartOptions(0,0,ChartGrid30 | ChartGrid70 );
r = StochK(14);
Plot( r, "StochK", colorBlack );
PlotOHLC( r,r,50,r, "", IIf( r > 50, colorRed, colorGreen ), styleCloud |
styleClipMinMax, 30, 70 );
```

### X-shift feature

The XShift parameter allows to displace (shift) the plot in horizontal direction by specified number of bars. This allows to plot displaced moving averages and projections into the future. See the following sample code of displaced moving average:

```
Periods = Param("Periods", 30, 2, 100 );
Displacement = Param("Displacement", 15, -50, 50 );

Plot( MA( C, Periods ), _DEFAULT_NAME(), ColorCycle, styleLine, 0, 0,
Displacement );
```

### PlotForeign() function

It is now easy to overlay price plots of multiple symbols using PlotForeign function:

```
PlotForeign( tickersymbol, name, color/barcolor, style = styleCandle | styleOwnScale, minvalue = {empty},
maxvalue = {empty}, xshift = 0)
```

Plots the foreign-symbol price chart (symbol is defined by *tickersymbol* parameter). Second argument *name* defines graph name used for displaying values in a title bar. Graph color could be static (if third argument is a number) or dynamic (when third argument is an array). Color indexes are related to the current palette (see Preferences/Color)

*style* defines chart plot style (see Plot() function for possible values)

```
PlotForeign( "^DJI", "Dow Jones", colorRed );
PlotForeign( "^NDX", "Nasdaq 100", colorBlue );
PlotForeign( "^IXIC", "Nasdaq Composite", colorGreen );
```

### Multiple plots using different scaling

Two new styles can be used to plot multiple graphs using different Y-scale: styleOwnScale and styleLeftAxisScale.

It also makes it easy to plot 2 or more "own scale" plots with the same scaling:

```
minimum = LastValue( Lowest( Volume ) );
maximum = LastValue( Highest( Volume ) );

Plot( Close, "Price", colorBlue, styleCandle );

/* two plots below use OwnScale but the scale is common because we
set min and max values of Y axis */
Plot( Volume, "Volume", colorGreen, styleHistogram | styleThick |
styleOwnScale, minimum, maximum );
Plot( MA( Volume, 15 ), "MA volume", colorRed, styleLine |
```



```
styleOwnScale, minimum, maximum );
```

New style: styleLeftAxisScale = 65536 – allows to plot more than one graph using common scaling but different from regular (right axis) scale.

Example: price plot plus volume and moving average plot:

```
// Plot price plot and its moving average
Plot( Close, "Price", colorWhite, styleCandle );
Plot( MA( Close, 20 ), "MAC", colorRed );

// Now plot volume and its moving average using left-hand axis
scaling
Plot( Volume , "Volume", colorBlue, styleLeftAxisScale |
styleHistogram | styleThick );
Plot( MA( Volume,15), "MAV", colorLightBlue, styleLeftAxisScale );
```

New parameters make it also easy to plot ribbons, for example:

```
Plot( Close, "Price", colorBlue, styleCandle );
Plot( 2, /* defines the height of the ribbon in percent of pane width
*/
"Ribbon",
IIf( up, colorGreen, IIf( down, colorRed, 0 )), /* choose color */
styleOwnScale|styleArea|styleNoLabel, -0.5, 100 );
```

### Using custom defined parameters

AmiBroker allows to create user-defined parameters. Such parameters are then available via [Parameters](#) dialog for quick and fast adjustment of indicator.

Most often used parameter functions are (click on the links to get more detailed description):

- [Param](#)( "name", default, min, max, steps, incr = 0 )
- [ParamStr](#)( "name", "default" );
- [ParamColor](#)( "name", defaultcolor );
- [ParamStyle](#)("name", defaultval = styleLine, mask = maskDefault )

They make it possible to define your own parameters in your indicators. Once Param functions are included in the formula you can right click over chart pane and select "Parameters" or press Ctrl+R, and change them via [Parameters](#) dialog and get immediate response.

The simplest case looks like this:

```
period = Param("RSI period", 12, 2, 50, 1 );
Plot( RSI( period ), "RSI( " + period + " )", colorRed );
```

Right click over the chart and choose "Parameters" and move the slider and you will see RSI plotted with different periods immediately as you move the slider.

Sample code below shows how to use [ParamStr](#) to get the ticker symbol and [ParamColor](#) to get colors.

```

ticker = ParamStr( "Ticker", "MSFT" );
sp = Param( "MA Period", 12, 2, 100 );
PlotForeign( ticker, "Chart of "+ticker,
             ParamColor( "Price Color", colorBlack ), styleCandle );
Plot( MA( Foreign( ticker, "C" ), sp ), "MA", ParamColor( "MA Color",
colorRed ) );

```

The following sample formula (from AmiBroker mailing list) that allows to visually align price peak/troughs with sine curve on the chart:

```

Cycle = Param( "Cycle Months", 12, 1, 12, 1 ) * 22; // 264 == 12mth, 22 == 1mth
xfactor = Param( "Stretch", 1, 0.1, 2, 0.1 ); // 1 == 1yr, 2 == 2yr
xshift = Param( "slide", 0, -22, 22, 2 ) / 3.1416^2; // slide curve 1 == 5days

x = 2 * 3.1416 / Cycle / xfactor;
y = sin( Cum(x) - xshift );

Plot( C, "Daily Chart", colorBlack, styleCandle | styleNoLabel );
Plot( y,
      "cycle =" + WriteVal( Cycle * xfactor / 22, 1.0 ) + "months",
      colorBlue, styleLine | styleNoLabel | styleOwnScale );

```

Right click over the chart and choose "Parameters" and move the sliders and you will see chart immediately reflecting your changes.

For more information on user-definable parameters please check also [Tutorial: Using drag-and-drop interface](#)

### Plotting texts at arbitrary positions on the chart

AmiBroker now allows annotation of the chart with text placed on any x, y position specified on the formula level using new [PlotText](#) function.

```
PlotText( "text", x, y, color, bkcolor = colorDefault )
```

where

x – is x-coordinate in bars (like in LineArray)

y – is y-coordinate in dollars

color is text color, bkcolor is background color. If bkcolor is NOT specified (or equal to colorDefault) text is written with TRANSPARENT background, any other value causes solid background with specified background color

Example:

```

Plot( C, "Price", colorBlack, styleLine );
Plot( MA( C, 20 ), "MA20", colorRed );

Buy = Cross( C, MA( C, 20 ) );
Sell = Cross( MA( C, 20 ), C );

dist = 1.5 * ATR( 10 );

```

```

for( i = 0; i < BarCount; i++ )
{
    if( Buy[i] ) PlotText( "Buy\n@" + C[ i ], i, L[ i ]-dist[i], colorGreen );
    if( Sell[i] ) PlotText( "Sell\n@" + C[ i ], i, H[ i ]+dist[i], colorRed,
colorYellow );
}

PlotShapes( Buy * shapeUpArrow + Sell * shapeDownArrow, IIf( Buy, colorGreen,
colorRed ) );

```

### Gradient fill of the background

AmiBroker 4.90 allows to fill indicator background with gradually changing color. To achieve this you need to use new function SetChartBkGradientFill( topcolor, bottomcolor, titlebkcolor = default )

The function enables background gradient color fill in indicators.

Please note that this is independent from chart background color (background color fills entire pane, gradient fill is only for actual chart interior, so axes area is not affected by gradient fill). Parameters are as follows:

topcolor – specifies top color of the gradient fill

bottomcolor – specifies bottom color of the gradient fill

titlebkcolor – (optional) the background color of title text. If not specified then top color is automatically used for title background.

Example:

```

SetChartBkGradientFill( ParamColor( "BgTop", colorWhite ), ParamColor( "BgBottom",
colorLightYellow ) );

```

### Miscellaneous

As you already know each plot has its own name that is used to create a title string which displays names and values of indicators. AmiBroker however allows you to override this automatic mechanism and define your own title string from the scratch. The **Title** reserved variable is used for that. You just assign a string to it and it will be displayed in the chart instead of automatically generated one.

Also there two more reserved variables (GraphXSpace and GraphZOrder) that allow to fine-tune indicator look.

They are all described in the table below.

Variable	Usage	Applies to
<b>Title</b>	<p>Defines title text</p> <p>If you use Title variable you have to specify colors in the string. Colors can be specified using \cXX sequence where XX is 2 digit number specifying color index \c38 – defines violet, there is a special sequence \c-1 that resets to default axis color. For example</p>	Indicators

	<p>Title = "This is written in \c38violet color \c27and this in green";</p> <p>You can also use new AFL function that makes it easier. Function is called EncodeColor( colornumber ).</p> <p>And you can write the above example like this:</p> <p>Title = "This is written in " + EncodeColor( colorViolet ) + "violet color " + EncodeColor( colorGreen ) + "and this in green";</p> <p>Multi-line caption is possible by simply embedding line break \n, for example: Title = "This is 1st line\nThis is second line";</p>	
<b>Tooltip</b>	<p>Allows you to define your own text for data tooltip</p> <p>Example:</p> <p>Tooltip = "This is my tool tip text showing close price: " + Close;</p>	Indicators
<b>GraphXSpace</b>	<p>defines how much extra space should be added above and below graph line (in percent).</p> <p>For example:</p> <p>GraphXSpace = 5;</p> <p>adds 5% extra space above and below the graph line. When GraphXSpace is not defined in the formula then default 2% is used.</p>	Indicators
<b>GraphZOrder</b>	<p>GraphZOrder variable allows to change the order of plotting indicator lines. When GraphZOrder is not defined or is zero (false) – old ordering (last to first) is used, when GraphZOrder is 1 (true) – reverse ordering is applied.</p>	Indicators

### Obsolete graph variables

This table shows obsolete reserved variables. They are still functional for backward-compatibility but new code should use Plot() functions only. What's more, when using new Plot() functions you should NOT use obsolete variables below.

Variable	Usage	Applies to
maxgraph	specifies maximum number of graphs to be drawn in custom indicator window (default=3)	Indicators
graphN	defines the formula for the graph number <i>N</i> (where <i>N</i> is a number 0,1,2,..., maxgraph-1)	Indicators
graphNopen, graphNhigh, graphNlow,	define additional O, H, L price arrays for candlestick and traditional bar charts	Indicators
graphNcolor	defines the color index of <i>N</i> th graph line. Color indexes are related to the current palette – see Preferences/Color.	Indicators

graphNbarcolor	defines the array that holds palette indexes for each bar drawn	Indicators
graphNstyle	defines the style of Mth graph. Style is defined as a combination (sum) of one or more following flags ( you can use predefined style__ constants instead of numbers)	Indicators

## How to create your own exploration

One of the most useful features of Automatic Analysis window is called "Exploration". Basically, an exploration works in a similar way to scan but instead of looking for and reporting just buy/sell signals it allows you to generate customizable screening (or exploration) report that can give you much more information than simple scan.

The idea behind an exploration is simple – one variable called **filter** controls which symbols/quotes are accepted. If "true" (or 1) is assigned to that variable for given symbol/quote it will be displayed in the report.

So, for example, the following formula will accept all symbols with closing prices greater than 50 :

```
filter = close > 50;
```

(NOTE: To create new formula please open [Formula Editor](#) using **Analysis-->Formula Editor** menu, type the formula and choose **Tools-->Send to Automatic Analysis** menu in Formula editor)

Note that exploration uses all range and filter settings that are also used by back-tester and scanning modes so you can get multiple signals (report lines) if you select "All quotations" range. To check just the most recent quote you should choose "**n last quotations**" and "**n=1**" as shown here:

Now, what about customizable reports?

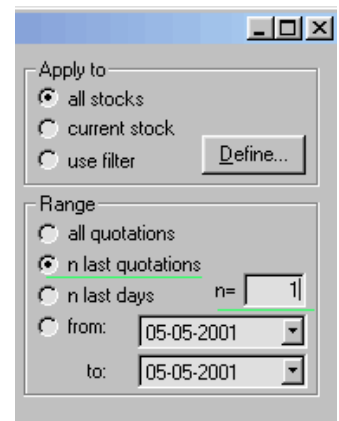
Yes, exploration mode allows you to create and then export a report with completely customizable columns and it is quite simple to do.

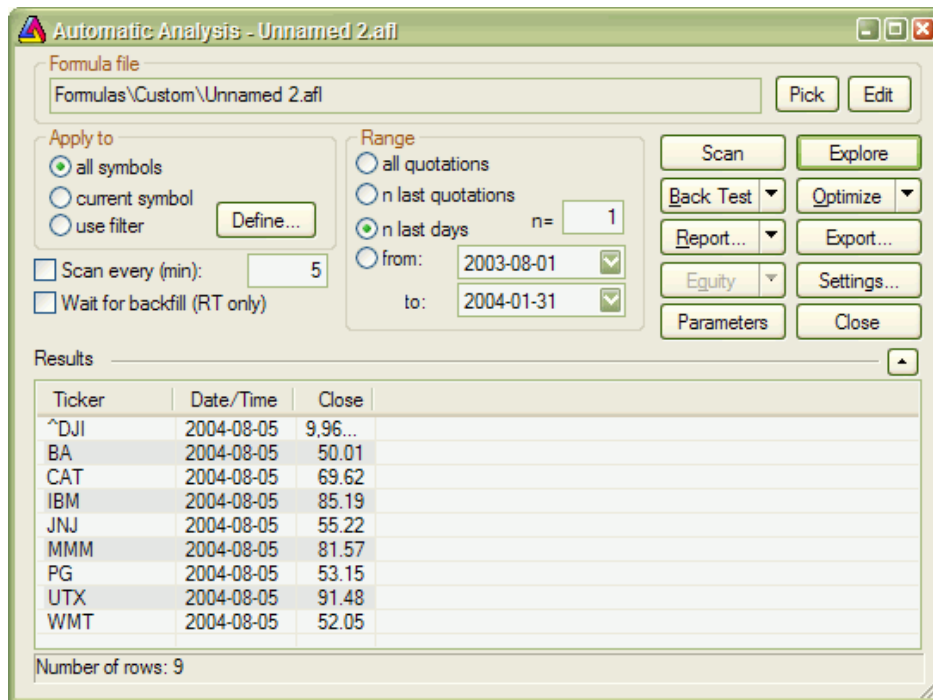
All you have to do is to tell AmiBroker what columns do you want. This can be done by calling [AddColumn](#) function in your exploration formula:

```
AddColumn( Close, "Close" );
```

The first argument of AddColumn function is the data ARRAY you want to display, the second argument defines the column caption

If you now press "**Explore**" button in Automatic Analysis window you will get the result similar to this:





Note that there are actually 3 columns: predefined Ticker and Date/Time column and one custom column holding close price. Note that only tickers with close price greater than 50 are reported.

Now you can click **"Export"** and your exploration will be saved to CSV (comma separated values) file that could be easily loaded to any other program including Excel for further analysis.

Actually AddColumn function accepts more arguments to allow you to customize the output even more. The full syntax is:

**AddColumn( array, name, format = 1.2, textColor = colorDefault, bkgndColor = colorDefault )**

**format** parameter allows you to define the formatting applied to numbers. By default all variables are displayed with 2 decimal digits, but you can change this by assigning a different value to this variable: 1.5 gives 5 decimal digits, 1.0 gives no decimal digits. So, in our example, typing:

```
AddColumn( Close, "Close", 1.4 );
```

will give closing prices displayed with 4 decimal digits.

(Note for advanced users: the integer part of this number can be used to pad formatted number with spaces – 6.0 will give no decimal digits but a number space-padded upto 6 characters.)

There are also special format pre-defined constants that allow to display date/time and single character codes:

- **formatDateTime** – produces date time formatted according to your system settings  
`AddColumn( DateTime(), "Date / Time", formatDateTime );`

- **formatChar** – allows outputting single ASCII character codes:  
 Example (produces signal file accepted by various other programs):  
`Buy=Cross(MACD(),Signal());`

```

Sell=Cross(Signal(), MACD());
Filter=Buy OR Sell;
SetOption("NoDefaultColumns", True );
AddColumn( DateTime(), "Date", formatDateTime );
AddColumn( IIf( Buy, 66, 83 ), "Signal", formatChar );

```

**textColor** and **bkgndColor** arguments allow you to produce colorful reports. By default result list is displayed using system color but you can override this behaviour providing your own colors.

For example, the code that displays close price in green color when 1 day rate of change is positive and otherwise uses red color:

```
AddColumn( Close, "Close", 1.4, IIF( ROC(C, 1 ) > 0, colorGreen, colorRed ) );
```

### Examples

The exploration mode is extremely flexible: you can, for example, export the whole database to CSV file using the following formula:

```

filter = 1; /* all symbols and quotes accepted */

AddColumn(Open,"Open",1.4);
AddColumn(High,"High",1.4);
AddColumn(Low,"Low",1.4);
AddColumn(Close,"Close",1.4);
AddColumn(Volume,"Volume",1.0);

```

This one will show you only heavily traded securities:

```

filter = volume > 5000000; /* adjust this threshold for your own
needs */
AddColumn(Close,"Close",1.4);
AddColumn(Volume,"Volume",1.0);

```

or...just show securities with volume being 30% above its 40-day exponential average

```

filter = volume > 1.3 * ema( volume, 40 );
AddColumn(Close,"Close",1.4);
AddColumn(Volume,"Volume",1.0);

```

With this one, you can export multiple indicator values for further analysis:

```

filter = close > ma( close, 20 ); /* only stocks trading above its 20
day MA*/
AddColumn( macd(), "MACD", 1.4 );
AddColumn( signal(), "Signal", 1.4 );
AddColumn( adx(), "ADX", 1.4 );
AddColumn( rsi(), "RSI", 1.4 );
AddColumn( roc( close, 15 ), "ROC(15)", 1.4 );
AddColumn( mfi(), "MFI", 1.4 );
AddColumn( obv(), "OBV", 1.4 );
AddColumn( cci(), "CCI", 1.4 );

```

```
AddColumn( ultimate(), "Ultimate", 1.4 );
```

One more example of color output:

```
Filter =1;

AddColumn( Close, "Close", 1.2 );
AddColumn( MACD(), "MACD", 1.4 , IIf( MACD() > 0, colorGreen,
colorRed ) );
AddTextColumn( FullName(), "Full name", 77 , colorDefault, IIf( Close
< 10, colorLightBlue, colorDefault ) );
```

### Final tip

Please don't forget that you can sort the results of the exploration by any column by simply clicking on its header.

## How to write your own chart commentary

One of the interesting aspects of using AmiBroker Formula Language is writing automatic chart commentaries. The idea behind this technique is as follows:

1. You write the commentary formula that consists of two basic elements: static texts and AFL expressions
2. AmiBroker evaluates expressions using currently selected symbol data and generates dynamic content
3. The mixture of static text and evaluated formulas are displayed in commentary output window
4. Additionally buy/sell arrows are plotted on the chart

Commentaries are available from *Analysis->Commentary* menu. When you open commentary window you will see two tabs: *Commentary* and *Formula*. In the *Formula* tab you can type the AFL statements which will be evaluated by AmiBroker resulting in dynamic commentary that appears in *Commentary* tab. The following sections will guide you through the steps needed to write your very own commentary formulas.

### Writing static texts

Static text elements written in the formula should be enclosed in the quotation marks and terminated by semicolon sign as shown below:

```
"This is sample static text statement";
```

You can write several statements and each statement will be placed in a new line in the commentary output window:

```
"This is first line of text";
"This is second line of text";
```

Please type these examples into edit field in the *Formula* tab and switch to *Commentary* tab. You will see the texts displayed in the output area but without any quotation marks or semicolons. This is because AmiBroker has evaluated this simple text statements into strings and it displayed the strings in the output window.



To write several lines of text you can use a couple of statements as shown above or you can do this using single statement and line break sequence ('\n'):

```
"This is first line of text\nThis is second line of text\nThis is
third line of text";
```

You can also concatenate the string constants which will result in single line text:

```
"This" +
" is" +
" single"+
" line" + " of text";
```

I guess that you are quite bored with these simple examples, let's start with some dynamic content.

### Dynamic content

To enable dynamic commentaries AFL has a couple of special functions available, but two of them are the most important: WriteVal() and WriteIF(). WriteIF() function is used for conditional text display and will be described later in this article, now let us see what we can do using WriteVal() function.

The AFL reference manual says:

<b>SYNTAX</b>	writeval( NUMBER ); writeval( ARRAY );
<b>RETURNS</b>	STRING
<b>FUNCTION</b>	This function can only be used within an Guru commentary. It is used to display the numeric value of NUMBER or ARRAY.

So, if you want to display a value of a number or currently selected bar of the array you should use writeval() function. But... wait a minute – what does it mean "currently selected bar of the array"? Let me explain this using simple formula (please type it in the *Formula* tab):

```
writeval( close );
```

When you switch to *Commentary* tab you will see the value of closing price (the same one which is displayed at the top of main price chart). But when you click on the chart in another place, selecting different date and then you click "Refresh" button you will see different value – the closing price at day you have selected. So writeval( close ) function displays the value of currently selected bar of close array. And it works exactly the same way with other arrays. If you write

```
writeval( macd() );
```

you will see the exact value of MACD indicator at the day you have selected in the main chart. Having our current know-how we are able to write some statistics:

```
"Closing price = " + WriteVal( close );
"Change since yesterday = " + WriteVal( close - ref( close, -1 ) );
"Percent chg. since yesterday = " + WriteVal( roc( close, 1 ) ) + "
%";
"MACD = " + WriteVal( macd() ) + " , Signal line = " + WriteVal(
```

```
signal() );
```

When you switch to *Commentary* tab you will see output similar to this one:

```
Closing price = 17.940
Change since yesterday = -0.180
Percent chg. since yesterday = -0.993 %
MACD = -0.001 , Signal line = 0.063
```

Quite nice, isn't it? You can also write current symbol ticker and selected date using `name()` and `date()` functions as shown below:

```
"Statistics of " + name() + " as of " + date();
```

But what we miss here is an ability to write something if some condition is met and write something different otherwise...

### Conditional text output

AFL is equipped with very nice function called `WriteIf()` that can output different texts depending on the condition. Let us look what documentation says:

<b>SYNTAX</b>	<code>writeif( EXPRESSION, "TRUE TEXT", "FALSE TEXT" )</code>
<b>RETURNS</b>	STRING
<b>FUNCTION</b>	This function can only be used within an Guru commentary. If <code>EXPRESSION</code> evaluates to "true", then the <code>TRUE TEXT</code> string is displayed within the commentary. If <code>EXPRESSION</code> evaluates to "false", then the <code>FALSE TEXT</code> string is displayed.

So we can easily output different text depending on expression, for example:

```
writeif( macd() > signal(), "The MACD is bullish because is is above
it's signal line", "The MACD is bearish because it is below its
signal line" );
```

You can also combine several `WriteIf()` function calls in order to handle more possibilities:

```
"The current market condition for "+ name() + " is: ";

avgcond1 = ( c > ema( close, 200) ) + 0.1 * ( close > ema( close, 90)
) + 0.1 * ( close > ema( close , 30 ) );
avgcond2 = -( c < ema( close, 200) ) - 0.1 * ( close < ema( close,
90) ) - 0.1 * ( close < ema( close , 30 ) );

WriteIf( avgcond1 == 1.2,
"Very Bullish",
WriteIf( avgcond1 == 1.1,
"Bullish",
WriteIf( avgcond1 == 1.0,
"Mildly Bullish", "" ) ) ) +
```

```
WriteIf( avgcond2 == -1.2,
"Very Bearish",
WriteIf( avgcond2 == -1.1,
"Bearish",
WriteIf( avgcond2 == -1.0,
"Mildly Bearish", "" ) ) );
```

The formula above will return the text "The current market condition for {your ticker here} is: Very Bullish" if close price is above 30 day average and close is above 90 day average and close is above 200 day average. In other cases the formula will give you Bullish, Mildly Bullish, Mildly Bearish, Bearish or Very Bearish ratings.

For more examples on AFL commentaries please check [AFL formula library](#) especially [MACD commentary](#) formula which demonstrates all techniques presented here.

Now you are ready to start with your own commentaries... Good luck!

## Using studies in AFL formulas

AmiBroker 3.52 introduces ability to reference hand-drawn studies from AFL formulas. This feature is quite unique among trading software and as you will find out using this feature is quite easy.

I will show you an example how to check if the trend line is broken from AFL code. All we need to do is three simple steps:

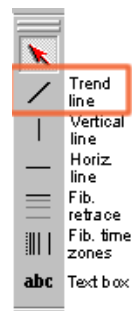
1. Draw a trend line
2. Define study ID
3. Write the formula that checks trend line break

### Drawing trend line

A trend line is a sloping line drawn between two prominent points on a chart.

In this example we will draw the rising trend line that defines the uptrend. This kind of trend line is usually drawn between two (or more) troughs (low points) to illustrate price support.

For sure you know how to draw a trend line in AmiBroker – just select a "Trend line" tool from "Draw" toolbar, find at least two recent troughs and just draw the line.

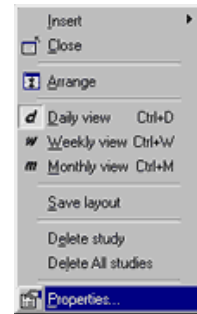


### Define study ID

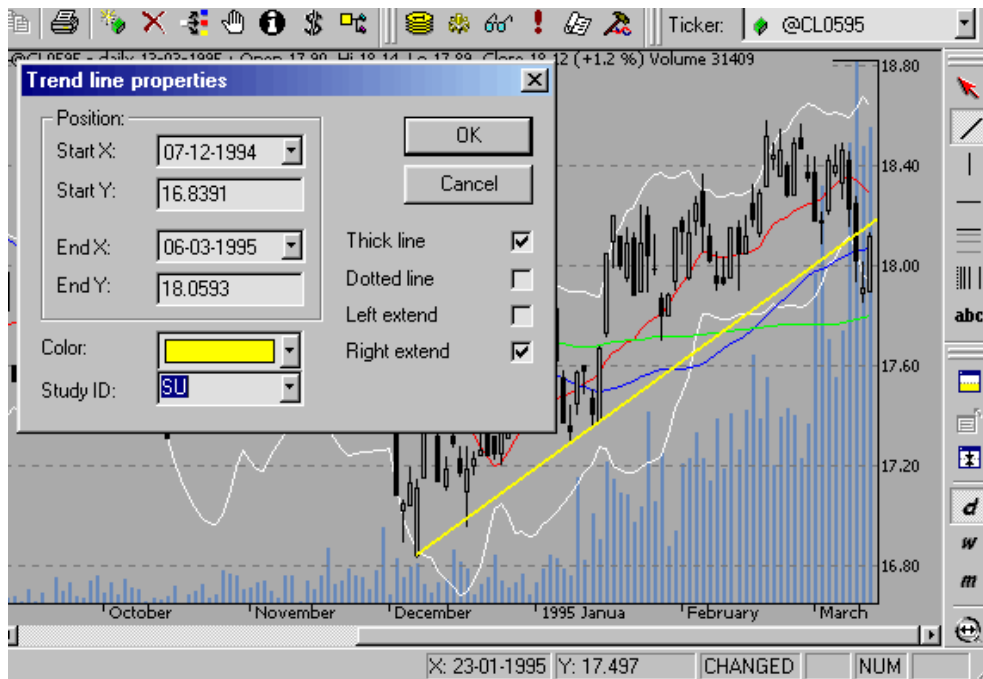
As you probably know, you can modify the properties of each line drawn in AmiBroker by clicking with the right mouse button over the study and selecting "Properties" from the menu. The properties dialog that shows up allows you to define exact start/end points and choose line colour, style and left and/or right extension mode.

For further analysis we will use the right-extended trend line (click on

appropriate checkbox) to make sure that the trend line is automatically extended when new data are added.



Since version 3.52 the properties dialog allows also to define "Study ID" (the combo below colour box). "Study ID" is a two-letter code of the study that can be assigned to any study within a chart that allows AmiBroker to reference it from AFL. Predefined identifiers are: "UP" – uptrend, "DN" – downtrend, "SU" – support, "RE" – resistance, "ST" – stop loss, however you can use ANY identifiers (there are no limitations except that AmiBroker accepts only 2 letter codes). This way if you draw the support lines in many symbols and give them all "SU" identifier then you will be able to reference the support line from AFL code.



So we will assign the "SU" study ID to the rising support trend line we have just drawn.

### Write the formula that checks trend line break

In this example we will detect if the closing price drops BELOW support trend line. This is actually very simple:

```
sell = cross( study( "SU" ), close, GetChartID() );
```

Note that study() function accepts two arguments: the first is StudyID two letter code that corresponds to one given in properties dialog; the second argument is chart ID – it should be taken either via GetChartID() function (then it refers to current indicator) or read from [Parameter dialog](#), Axes & Grid: Miscellaneous: Chart ID.

## Back-testing your trading ideas

### Introduction

One of the most useful things that you can do in [automatic analysis window](#) is to back-test your trading strategy on historical data. This can give you valuable insight into strengths and weak points of your system **before** investing real money. This single AmiBroker feature is can save lots of money for you.

### Writing your trading rules

First you need to have objective (or mechanical) rules to enter and exit the market. This step is the base of your strategy and you need to think about it yourself since the system must match your risk tolerance, portfolio size, money management techniques, and many other individual factors.

Once you have your own rules for trading you should write them as buy and sell rules in AmiBroker Formula Lanugage (plus short and cover if you want to test also short trading).

In this chapter we will consider very basic moving average cross over system. The system would buy stocks/contracts when close price rises above 45-day exponential moving average and will sell stocks/contracts when close price falls below 45-day exponential moving average.

The exponential moving average can be calculated in AFL using its built-in function EMA. All you need to do is to specify the input array and averaging period, so the 45-day exponential moving average of closing prices can be obtained by the following statement:

```
ema( close, 45 );
```

The **close** identifier refers to built-in array holding closing prices of currently analysed symbol.

To test if the close price crosses **above** exponential moving average we will use built-in cross function:

```
buy = cross( close, ema( close, 45 ) );
```

The above statement defines a buy trading rule. It gives "1" or "true" when close price crosses above ema( close, 45 ). Then we can write the sell rule which would give "1" when opposite situation happens – close price crosses **below** ema( close, 45 ):

```
sell = cross( ema( close, 45 ), close );
```

Please note that we are using **the same** cross function but the **opposite** order of arguments.

So complete formula for long trades will look like this:

```
buy = cross( close, ema( close, 45 ) );
sell = cross( ema( close, 45 ), close );
```

**NOTE:** To create new formula please open [Formula Editor](#) using **Analysis->Formula Editor** menu, type the formula and choose **Tools->Send to Automatic Analysis** menu in Formula editor

### Back testing

To back-test your system just click on the **Back test** button in the Automatic analysis window. Make sure you have typed in the formula that contains at least buy and sell trading rules (as shown above). When the formula is correct AmiBroker starts analysing your symbols according to your trading rules and generates a list of simulated trades. The whole process is very fast – you can back test thousands of symbols in a matter of minutes. The progress window will show you estimated completion time. If you want to stop the process you can just click Cancel button in the progress window.

### Analysing results

When the process is finished the list of simulated trades is shown in the bottom part of Automatic analysis window. (the **Results** pane). You can examine when the buy and sell signals occurred just by double clicking on the trade in **Results** pane. This will give you raw or unfiltered signals for every bar when buy and sell conditions are met. If you want to see only single trade arrows (opening and closing currently selected trade) you should double click the line while holding SHIFT key pressed down. Alternatively you can choose the type of display by selecting appropriate item from the context menu that appears when you click on the results pane with a right mouse button.

In addition to the results list you can get very detailed statistics on the performance of your system by clicking on the **Report** button. To find out more about report statistics please check out [report window description](#).

### Changing your back testing settings

Back testing engine in AmiBroker uses some predefined values for performing its task including the portfolio size, periodicity (daily/weekly/monthly), amount of commission, interest rate, maximum loss and profit target stops, type of trades, price fields and so on. All these settings could be changed by the user using [settings window](#). After changing settings please remember to run your back testing again if you want the results to be in-sync with the settings.

For example, to back test on weekly bars instead of daily just click on the **Settings** button select **Weekly** from **Periodicity** combo box and click **OK**, then run your analysis by clicking **Back test**.

### Reserved variable names

The following table shows the names of reserved variables used by Automatic Analyser. The meaning and examples on using them are given later in this chapter.

Variable	Usage	Applies to
buy	defines "buy" (enter long position) trading rule	Automatic Analysis, Commentary
sell	defines "sell" (close long position) trading rule	Automatic Analysis, Commentary
short	defines "short" (enter short position – short sell) trading rule	Automatic Analysis
cover	defines "cover" (close short position – buy to cover) trading rule	Automatic Analysis
buyprice	defines buying price array (this array is filled in with the default values according to the Automatic Analyser settings)	Automatic Analysis

sellprice	defines selling price array (this array is filled in with the default values according to the Automatic Analyser settings)	Automatic Analysis
shortprice	defines short selling price array (this array is filled in with the default values according to the Automatic Analyser settings)	Automatic Analysis
coverprice	defines buy to cover price array (this array is filled in with the default values according to the Automatic Analyser settings)	Automatic Analysis
exclude	If defined, a true (or 1) value of this variable excludes current symbol from scan/exploration/back test. They are also not considered in buy and hold calculations. Useful when you want to narrow your analysis to certain set of symbols.	Automatic Analysis
roundlotsize	defines round lot sizes used by backtester (see explanations below)	Automatic Analysis (new in 4.10)
ticksize	defines tick size used to align prices generated by <b><i>built-in stops</i></b> (see explanations below) (note: it does not affect entry/exit prices specified by buyprice/sellprice/shortprice/coverprice)	Automatic Analysis (new in 4.10)
pointvalue	allows to read and modify future contract point value (see <a href="#">backtesting futures</a> )	Automatic Analysis (new in 4.10)
margindeposit	allows to read and modify future contract margin (see <a href="#">backtesting futures</a> )	Automatic Analysis (new in 4.10)
positionsiz	Allows control dollar amount or percentage of portfolio that is invested into the trade (see explanations below)	Automatic Analysis (new in 3.9)

### Advanced concepts

Until now we discussed fairly simple use of the back tester. AmiBroker, however supports much more sophisticated methods and concepts that will be discussed later on in this chapter. Please note that the beginner user should first play a little bit with the easier topics described above before proceeding.

So, when you are ready, please take a look at the following recently introduced features of the back-tester:

- a) AFL scripting host for advanced formula writers
- b) enhanced support for short trades
- c) the way to control order execution price from the script
- d) various kinds of stops in back tester
- e) position sizing
- f) round lot size and tick size
- g) margin account
- h) [backtesting futures](#)

AFL scripting host is an advanced topic that is covered in a separate document available [here](#) and I won't discuss it in this document. Remaining features are much more easy to understand.

### Short trade support

In the previous versions of AmiBroker, if you wanted to back-test system using both long and short trades, you could only simulate stop-and-reverse strategy. When long position was closed a new short position was opened immediately. It was because buy and sell reserved variables were used for both types of trades.

Now (with version 3.59 or higher) there are separate reserved variables for opening and closing long and short trades:

buy – "true" or 1 value opens long trade  
 sell – "true" or 1 value closes long trade  
 short – "true" or 1 value opens short trade  
 cover – "true" or 1 value closes short trade

Som in order to back-test short trades you need to assign short and cover variables.

If you use stop-and-reverse system (always on the market) simply assign sell to short and buy to cover

```
short = sell;
cover = buy;
```

This simulates the way pre-3.59 versions worked.

But now AmiBroker enables you to have separate trading rules for going long and for going short as shown in this simple example:

```
// long trades entry and exit rules:
buy = cross( cci(), 100 );
sell = cross( 100, cci() );

// short trades entry and exit rules:
short = cross( -100, cci() );
cover = cross( cci(), -100 );
```

Note that in this example if CCI is between -100 and 100 you are out of the market.

### Controlling trade price

AmiBroker now provides 4 new reserved variables for specifying the price at which buy, sell, short and cover orders are executed. These arrays have the following names: buyprice, sellprice, shortprice and coverprice.

The main application of these variables is controlling trade price:

```
BuyPrice = IIF( dayofweek() == 1, HIGH, CLOSE );
// on monday buy at high, otherwise buy on close
```

So you can write the following to simulate real stop-orders:

```
BuyStop = ... the formula for buy stop level;
SellStop = ... the formula for sell stop level;

// if anytime during the day prices rise above buystop level
(high>buystop)
// the buy order takes place (at buystop or low whichever is higher)
Buy = Cross( High, BuyStop );
```



```
// if anytime during the day prices fall below sellprice level ( low
< sellstop )
// the sell order takes place (at sellstop or high whichever is
lower)
Sell = Cross( SellPrice, SellStop);

BuyPrice = max( BuyStop, Low ); // make sure buy price not less than
Low
SellPrice = min( SellStop, High ); // make sure sell price not
greater than High
```

Please note that AmiBroker presets buyprice, sellprice, shortprice and coverprice array variables with the values defined in system test settings window (shown below), so you can but don't need to define them in your formula. If you don't define them AmiBroker works as in the old versions.

During back-testing AmiBroker will check if the values you assigned to buyprice, sellprice, shortprice, coverprice fit into high-low range of given bar. If not, AmiBroker will adjust it to high price (if price array value is higher than high) or to the low price (if price array value is lower than low)

**Backtester settings**

General Trades Stops Report Portfolio

**General settings**

Initial equity: 10000 ☒ Allow position size shrinking

Positions: Long ☐ Activate stops immediately (when turned on, stops are checked AFTER current bar signals)

Periodicity: Daily ☐ Reverse entry signal forces exit

Min. shares: 0.0001 ☐ Allow same bar exit (single bar trade)

Min. pos. value: 0 ☐ Futures mode

☐ Pad and align all data to reference symbol: ^RUT (turning this on may slightly change indicators if you have data holes)

**Defaults**

Round lot size: 0 (zero means allow fractional # of shares)

Tick size: 0 (zero means no minimum change)

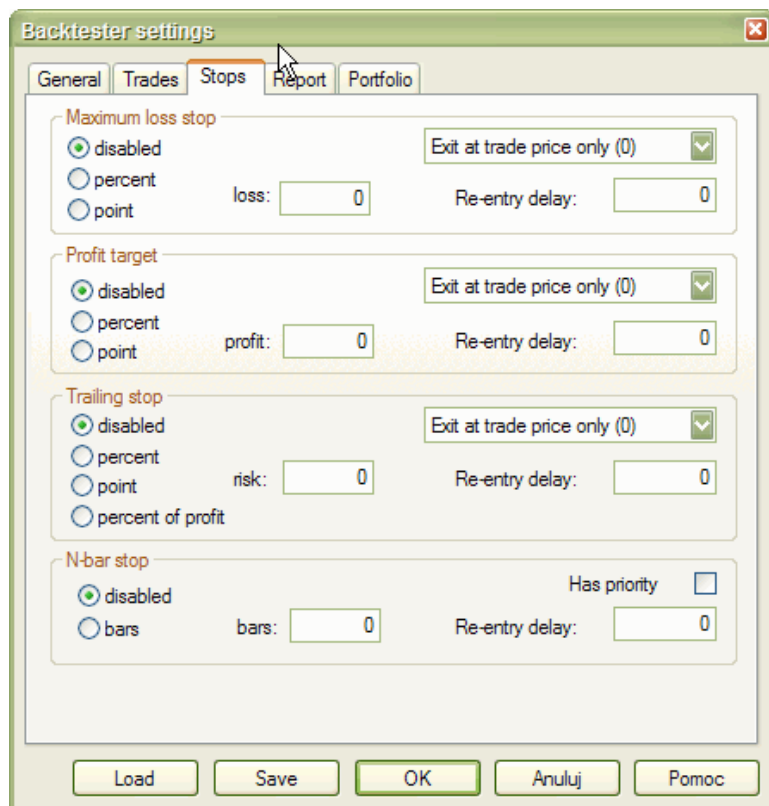
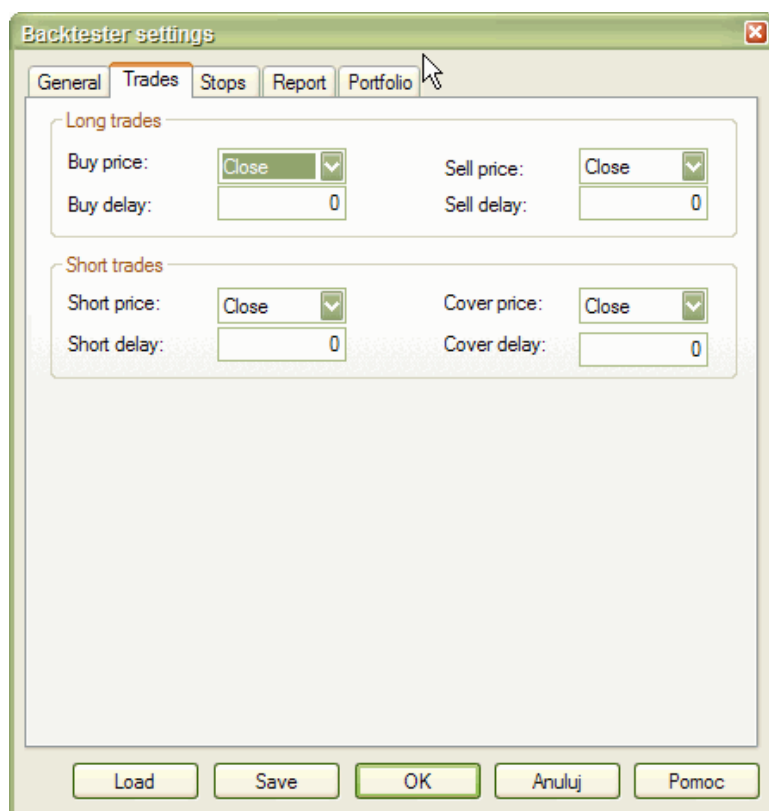
**Commissions & rates**

☐ commission table ☐ percent ☐ \$ per trade ☒ \$ per share/contract

Define... Annual interest rate: 0 Account margin: 100 (100 means no margin account)

0.01

Load Save OK Anuluj Pomoc



### Profit target stops

As you can see in the picture above, new settings for profit target stops are available in the system test

settings window. Profit target stops are executed when the **high** price for a given day exceeds the stop level that can be given as a percentage or point increase from the buying price. By default stops are executed at price that you define as sell price array (for long trades) or cover price array (for short trades). This behaviour can be changed by using "Exit at stop" feature.

### "Exit at stop" feature

If you mark "Exit at stop" box in the settings the stops will be executed at exact stop level, i.e. if you define profit target stop at +10% your stop and the buy price was 50 stop order will be executed at 55 even if your sell price array contains different value (for example closing price of 56).

Maximum loss stops work in a similar manner – they are executed when the **low** price for a given day drops below the stop level that can be given as a percentage or point increase from the buying price

### Trailing stops

This kind of stop is used to protect profits as it tracks your trade so each time a position value reaches a new high, the trailing stop is placed at a higher level. When the profit drops below the trailing stop level the position is closed. This mechanism is illustrated in the picture below (10% trailing stop is shown):



The trailing stop, as well as two other kind of stops could be enabled from user interface (Automatic analysis' [Settings window](#)) or from the formula level – using ApplyStop function:

To reproduce the example above you would need to add the following code to your automatic analysis formula:

```
ApplyStop( 2, 1, 10, 1 ); // 10% trailing stop, percent mode, exit at stop ON
```

or you can write it using predefined constants that are more descriptive

```
ApplyStop( stopTypeTrail, stopModePercent, 10, True );
```

Trailing stops could be also defined in points (dollars) and percent of profit (risk). In the latter case the amount parameter defines the percentage of profits that could be lost without activating the stop. So 20% percent of profit (risk) stop will exit your trade that has maximum profit of \$100 when the profit decreases below \$80.

### Dynamic stops

The ApplyStop() function allows now to change the stop level from trade to trade. This enables you to implement for example volatility-based stops very easily.

For example to apply maximum loss stop that will adapt the maximum acceptable loss based on 10 day average true range you would need to write:

```
ApplyStop( 0, 2, 2 * ATR( 10 ), 1 );
```

or you can write it using predefined constants that are more descriptive

```
ApplyStop( stopTypeLoss, stopModePoint, 2 * ATR( 10 ), True );
```

The function above will place the stop 2 times 10 day ATR below entry price.

As ATR changes from trade to trade – this will result in dynamic, volatility based stop level. Please note that 3rd parameter of ApplyStop function (the amount) is sampled at the trade entry and held throughout the trade. So in the example above it uses ATR(10) value from the date of the entry. Further changes of ATR do not affect the stop level.

See complete [APPLYSTOP](#) function documentation for more details.

### Coding your own custom stop types

ApplyStop function is intended to cover most "popular" kinds of stops. You can however code your own kind of stops and exits using looping code. For example the following re-implements profit target stop and shows how to refer to the trade entry price in your formulas:

```
/* a sample low-level implementation of Profit-target stop in AFL: */

Buy = Cross( MACD(), Signal() );

priceatbuy=0;

for( i = 0; i < BarCount; i++ )
{
    if( priceatbuy == 0 &Buy[ i ] )
        priceatbuy = BuyPrice[ i ];

    if( priceatbuy > 0 &SellPrice[ i ] > 1.1 * priceatbuy )
    {
        Sell[ i ] = 1;
        SellPrice[ i ] = 1.1 * priceatbuy;
        priceatbuy = 0;
    }
}
```

```

else
    sell[ i ] = 0;
}

```

## Position sizing

This is a new feature in version 3.9. Position sizing in backtester is implemented by means of new reserved variable

PositionSize = <size array>

Now you can control dollar amount or percentage of portfolio that is invested into the trade

- positive number define (dollar) amount that is invested into the trade for example:

```
PositionSize = 1000; // invest $1000 in every trade
```

- negative numbers -100..-1 define percentage:  
-100 gives 100% of current portfolio size,  
-33 gives 33% of available equity for example:

```
PositionSize = -50; /* always invest only half of the current equity */
```

- dynamic sizing example:

```
PositionSize = - 100 + RSI();
```

as RSI varies from 0..100 this will result in position depending on RSI values → low values of RSI will result in higher percentage invested

If less than 100% of available cash is invested then the remaining amount earns interest rate as defined in the settings.

There is also a new checkbox in the AA settings window: "Allow position size shrinking" – this controls how backtester handles the situation when requested position size (via PositionSize variable) exceeds available cash: when this flag is checked the position is entered with size shinked to available cash if it is unchecked the position is not entered.

To see actual position sizes please use a new report mode in AA settings window: "Trade list with prices and pos. size"

For the end, here is an example of Tharp's ATR-based position sizing technique coded in AFL:

```

Buy = <your buy formula here>
Sell = 0; // selling only by stop

TrailStopAmount = 2 * ATR( 20 );
Capital = 100000; /* IMPORTANT: Set it also in the Settings: Initial Equity */

Risk = 0.01*Capital;
PositionSize = (Risk/TrailStopAmount)*BuyPrice;
ApplyStop( 2, 2, TrailStopAmount, 1 );

```

The technique could be summarized as follows:

The total equity per symbol is \$100,000, we set the risk level at 1% of total equity. Risk level is defined as follows: if a trailing stop on a \$50 stock is at, say, \$45 (the value of two ATR's against the position), the \$5 loss is divided into the \$1000 risk to give 200 shares to buy. So, the loss risk is \$1000 but the allocation risk is 200 shares x \$50/share or \$10,000. So, we are allocating 10% of the equity to the purchase but only risking \$1000. (*Edited excerpt from the AmiBroker mailing list*)

## Round lot size and tick size

### Round lot size

Various instruments are traded with various "trading units" or "blocks". For example you can purchase fractional number of units of mutual fund, but you can not purchase fractional number of shares. Sometimes you have to buy in 10s or 100s lots. AmiBroker now allows you to specify the block size on global and per-symbol level.

You can define per-symbol round lot size in the Symbol->Information page (pic. 3). The value of zero means that the symbol has no special round lot size and will use "Default round lot size" (global setting) from the Automatic Analysis settings page (pic. 1). If default size is set also to zero it means that fractional number of shares/contracts are allowed.

You can also control round lot size directly from your AFL formula using RoundLotSize reserved variable, for example:

```
RoundLotSize = 10;
```

### Tick size

This setting controls the minimum price move of given symbol. You can define it on global and per-symbol level. As with round lot size, you can define per-symbol tick size in the Symbol->Information page (pic. 3). The value of zero instructs AmiBroker to use "default tick size" defined in the Settings page (pic. 1) of Automatic Analysis window. If default tick size is also set to zero it means that there is no minimum price move.

You can set and retrieve the tick size also from AFL formula using TickSize reserved variable, for example:

```
TickSize = 0.01;
```

Note that the tick size setting affects **ONLY** trades exited by built-in stops and/or ApplyStop(). The backtester assumes that price data follow tick size requirements and it does not change price arrays supplied by the user.

So specifying tick size makes sense only if you are using built-in stops so exit points are generated at "allowed" price levels instead of calculated ones. For example in Japan – you can not have fractional parts of yen so you should define global ticksize to 1, so built-in stops exit trades at integer levels.

## Margin account

*Account margin* setting defines percentage margin requirement for **entire account**. The default value of *Account margin* is 100. This means that you have to provide 100% funds to enter the trade, and this is the

way how backtester worked in previous versions. But now you can simulate a margin account. When you buy on margin you are simply borrowing money from your broker to buy stock. With current regulations you can put up 50% of the purchase price of the stock you wish to buy and borrow the other half from your broker. To simulate this just enter 50 in the *Account margin* field (see pic. 1) . If your initial equity is set to 10000 your buying power will be then 20000 and you will be able to enter bigger positions. Please note that this settings sets the margin for entire account and it is NOT related to futures trading at all. In other words you can trade stocks on margin account.

### **Additional settings**

- "Reverse entry signal forces exit" check box to the Backtester settings.  
When it is ON (the default setting) – backtester works as in previous versions and closes already open position if new entry signal in reverse direction is encountered. If this switch is OFF – even if reverse signal occurs backtester maintains currently open trade and does not close position until regular exit (sell or cover) signal is generated.  
In other words when this switch is OFF backtester ignores Short signals during long trades and ignores Buy signals during short trades.
- "Allow same bar exit (single bar trade)" option to the Settings  
When it is ON (the default settings) – entry and exit at the very same bar is allowed (as in previous versions)  
if it is OFF – exit can happen starting from next bar only (this applies to regular signals, there is a separate setting for ApplyStop-generated exits). Switching it to OFF allows to reproduce the behaviour of MS backtester that is not able to handle same day exits.
- "Activate stops immediately"

This setting solves the problem of testing systems that enter trades on market open. In versions prior to 4.09 backtester assumed that you were entering trades on market close so built-in stops were activated from the next day. The problem was when you in fact defined open price as the trade entry price – then same day price fluctuations did not trigger the stops. There were some published workarounds based on AFL code but now you don't need to use them. Simply if you trade on open you should mark "Activate stops immediately" (pic. 1).

You may ask why do not simply check the buyprice or shortprice array if it is equal to open price. Unfortunately this won't work. Why? Simply because there are doji days when open price equals close and then backtester will never know if trade was entered at market open or close. So we really need a separate setting.

### **See Also:**

[Portfolio-level backtesting](#) article.

[Backtesting systems for futures contracts](#) article.

[APPLYSTOP](#) function description

[Using AFL editor](#) section of the guide.

[Insider guide to backtester](#) (newsletter 1/2002)

## Portfolio-level backtesting

**IMPORTANT:** Please read first Tutorial: [Backtesting your trading ideas article](#)

New backtester **works on PORTFOLIO LEVEL**, it means that there is single portfolio equity and position sizing refers to portfolio equity. Portfolio equity is equal to available cash plus sum of all simultaneously open positions at given time.

AmiBroker's **portfolio backtester** lets you combine trading signals and trade sizing strategies into simulations which exactly mimic the way you would trade in real time. A core feature is its ability to perform dynamic money management and risk control at the portfolio level. Position sizes are determined with full knowledge of what's going on at the portfolio level at the moment the sizing decision is made. Just like you do in reality.

### HOW TO SET IT UP ?

**There are only two things that need to be done to perform portfolio backtest**

1. You need to have first the formula that generates buy / sell / short /cover signals as described in ["Backtesting your trading ideas"](#) article
2. You should define how many simultaneous trades you want to test and what position sizing algorithm you want to use.

### SETTING UP MAXIMUM NUMBER OF SIMULTANEOUSLY OPEN TRADES

There are two ways to set the maximum number of simultaneously open trades:

1. Go to the **Settings** dialog, switch to **Portfolio** tab and enter the number to **Max. Open Positions** field
2. Define the maximum in the formula itself (this overrides any setting in the Settings window) using [SetOption](#) function:

```
SetOption("MaxOpenPositions", 5 ); // This sets maximum number of open positions to 5
```

### SETTING UP POSITION SIZE

**IMPORTANT:** to enable more than one symbol to be traded you have to add PositionSize variable to your formula, so less than 100% of funds are invested in single security:

```
PositionSize = -25; // invest 25% of portfolio equity in single trade
```

or

```
PositionSize = 5000; // invest $5000 into single trade
```

There is a quite common way of setting both position size and maximum number of open positions so equity is spread equally among trades:

```
PosQty = 5; // You can define here how many open positions you want
SetOption("MaxOpenPositions", PosQty );
```



```
PositionSize = -100/PosQty; // invest 100% of portfolio equity divided by max.
position count
```

You can also use more sophisticated position sizing methods. For example volatility-based position sizing (Van Tharp-style):

```
PositionSize = -2 * BuyPrice/(2*ATR(10));
```

That way you are investing investing 2% of PORTFOLIO equity in the trade adjusted by BuyPrice/2\*ATR factor.

### **USING POSITION SCORE**

You can use new PositionScore variable to decide which trades should be entered if there are more entry signals on different securities than maximum allowable number of open positions or available funds. In such case AmiBroker will use the absolute value of PositionScore variable to decide which trades are preferred. See the code below. It implements simple MA crossover system, but with additional flavour of preferring entering trades on symbols that have low RSI value. If more buy signals occur than available cash/max. positions then the stock with lower RSI will be preferred. You can watch selection process if you backtest with "Detailed log" report mode turned on.

The code below includes also the example how to find optimum number of simultaneously open positions using new Optimization in Portfolio mode.

```

/*****
** REGULAR PORTFOLIO mode
** This sample optimization
** finds what is optimum number of positions open simultaneously
**
*****/

SetOption("InitialEquity", 20000 );
SetTradeDelays(1,1,1,1);
RoundLotSize = 1;

posqty = Optimize("PosQty", 4, 1, 20, 1 );
SetOption("MaxOpenPositions", posqty);

// desired position size is 100% portfolio equity
// divided by PosQty positions

PositionSize = -100/posqty;

// The system is very simple...
// MA parameters could be optimized too...
p1 = 10;
p2 = 22;
// simple MA crossover
Short=Cross( MA(C,p1) , MA(C,p2) );
Buy=Cross( MA(C,p2) , MA(C,p1) );
// always in the market
Sell=Short;
```

```
Cover=Buy;  
  
// now additional score  
// that is used to rank equities  
// when there are more ENTRY signals that available  
// positions/cash  
PositionScore = 100-RSI(); // prefer stocks that have low RSI;
```

## **BACKTEST MODES**

AmiBroker 5.0 offers 4 different backtest modes:

- regular mode (backtestRegular)
- regular raw mode (backtestRegularRaw)
- regular raw + multiple positions mode (backtestRegularRawMulti)
- rotational trading mode (backtestRotational)

All "regular" modes use buy/sell/short/cover signals to enter/exit trades, while "rotational" mode (aka "ranking / switching" system) uses only position score and is descibed later.

Backtest modes are switchable using [SetBacktestMode\(\)](#) AFL function.

The difference between "regular" modes is how repeated (also known as "redundant" or "extra") entry signals are handled. An "extra" entry signal is the signal that comes AFTER initial entry but before first matching exit signal.

In the regular mode – the default one, redundant entry signals are removed as shown in the picture below.

RAW SIGNALS															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
AAPL	Sell	Buy	Buy	Buy	Sell	Sell	Buy	Buy	Buy	Buy	Buy	Buy	Buy	Sell	Sell
MSFT	Sell	Sell	Buy	Sell	Sell	Buy	Sell	Sell	Sell	Sell	Sell	Buy	Sell	Sell	Sell
INTC	Buy	Buy	Buy	Buy	Buy	Buy	Buy	Buy	Buy	Buy	Buy	Sell	Sell	Sell	Sell
CSCO	Buy	Buy	Buy	Sell	Sell	Sell	Buy	Buy	Buy	Buy	Sell	Sell	Sell	Buy	Buy
LVL	Sell	Buy	Buy	Buy	Buy	Sell	Sell	Sell	Sell	Buy	Buy	Buy	Sell	Sell	Sell
AMGN	Buy	Buy	Sell	Sell	Buy	Buy	Buy	Sell	Sell	Buy	Buy	Buy	Buy	Sell	Sell
PHASE 1 - POTENTIAL TRADES - MATCHING BUY WITH SELL SIGNALS - EXTRA SIGNALS REMOVED															
Numbers in parentheses mean the POSITION SCORE at entry signal															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
AAPL		Buy(10)			Sell		Buy(8)							Sell	
MSFT			Buy(6)	Sell		Buy(1)	Sell					Buy(1)	Sell		
INTC	Buy(3)											Sell			
CSCO	Buy(5)			Sell			Buy(10)				Sell			Buy(1)	
LVL		Buy(8)				Sell				Buy(3)			Sell		
AMGN	Buy(4)		Sell		Buy(3)			Sell		Buy(2)				Sell	
PHASE 2 - PICKING TOP TRADES - MAX OPEN POS = 2, TRADES PICKED HAVE HIGHEST SCORE, ONCE PICKED, REMAIN IN PLACE															
GREY COLOR MEANS SKIPPED TRADE															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
AAPL		Buy(10)			Sell		Buy(8)							Sell	
MSFT			Buy(6)	Sell		Buy(1)	Sell					Buy(1)	Sell		
INTC	Buy(3)											Sell			
CSCO	Buy(5)			Sell			Buy(10)				Sell			Buy(1)	
LVL		Buy(8)				Sell				Buy(3)			Sell		
AMGN	Buy(4)		Sell		Buy(3)			Sell		Buy(2)				Sell	

As you can see Buy-Sell signal pairs are matched and treated as a TRADE. If trade is NOT entered on first entry signal due to weak rank, not enough cash or reaching the maximum open position count, subsequent entry signals are ignored until matching exit signal. After exit signal, the next entry signal will be possible candidate for entering trade. The process of removing excess signals occurring after first buy and matching sell (and short-cover pair respectively) is the same as `ExRem()` AFL function provides. To use regular mode you don't need to call `SetBacktestMode` function at all, as this is the default mode.

You may or may not consider removing extra signals desirable. If you want to act on ANY entry signal you need to use second mode – `backtestRegularRaw`. To turn it on you need to include this line in the code:

```
// signal-based backtest, redundant (raw) signals are NOT removed, only one
position per symbol allowed
SetBacktestMode( backtestRegularRaw );
```

It does NOT remove redundant entry signals and will act on ANY entry provided that it is scored highly enough and there is a cash available and maximum number of open positions is not reached. It will however allow only ONE OPEN POSITION per symbol at any given time. It means that if log trade is already open and later in the sequence appears an extra buy signal, it will be ignored until a "sell" signal comes (short-cover signals work the same). Note that you can still use `sigScaleIn/sigScaleOut` to increase or decrease the size of this existing position, but it will appear as single line in backtest result list.

If you want ALL repeated entry signals to be acted and allow to open multiple, separate positions on the same

symbol without scaling in/out effect (so multiple positions on the same symbol open simultaneously appear as separate lines in the backtest report) you need to use `backtestRegularRawMulti` mode by adding the following line to the code:

```
SetBacktestMode( backtestRegularRawMulti );
```

In this mode MULTIPLE positions per symbol will be open if BUY/SHORT signal is "true" for more than one bar and there are free funds. Sell/Cover exit all open positions on given symbol, Scale-In/Out work on all open positions of given symbol at once.

## ROTATIONAL TRADING

Rotational trading (also known as fund-switching or scoring and ranking) is possible too. For more information see the description of [EnableRotationalTrading](#) function.

## HOLDMINBARS and EARLY EXIT FEES

(Note that these features are available in portfolio-backtester only and not compatible with old backtester or `Equity()` function)

HoldMinBars is a feature that disables exit during user-specified number of bars even if signals/stops are generated during that period

Please note that IF during HoldMinBars period ANY stop is generated it is ignored. Also this period is ignored when it comes to calculation of trailing stops (new highest highs and drops below trailing stops generated during HoldMinBars are ignored). This setting, similar to `EarlyExitFee/EarlyExitBars` is available on per-symbol basis (i.e. it can be set to different value for each symbol)

Example:

```
SetOption( "HoldMinBars", 127 );
Buy=BarIndex() == 0;
Sell=1;
// even if sell signals are generated each day,
// they are ignored until bar 128
```

Early exit (redemption) fee is charged when trade is exited during first N bars since entry.

The fee is added to exit commission and you will see it in the commissions reported for example in detailed log. However, it is NOT reflected in the portfolio equity unless trade really exits during first N bars – this is to prevent affecting drawdowns if trade was NOT exited early.

```
// these two new options can be set on per-symbol basis
// how many bars (trading days)
// an early exit (redemption) fee is applied
SetOption( "EarlyExitBars", 128 );
// early redemption fee (in percent)
SetOption( "EarlyExitFee", 2 );
```

(note 180 calendar days is 128 or 129 trading days)

```
// how to set it up on per-symbol basis?
// it is simple - use 'if' statement
if( Name() == "SYMBOL1" )
```

```

{
    SetOption("EarlyExitBars", 128 );
    SetOption("EarlyExitFee", 2 );
}

if( Name() == "SYMBOL2" )
{
    SetOption("EarlyExitBars", 25 );
    SetOption("EarlyExitFee", 1 );
}

```

In addition to HoldMinBars, EarlyExitBars there are sibling features (added in 4.90) called **HoldMinDays** and **EarlyExitDays** that work with **calendar days** instead of data bars. So we can rewrite previous examples to use calendar days accurately:

```

// even if sell signals are generated each day,
// they are ignored until 180 calendar days since entry
SetOption("HoldMinBars", 180 );
Buy=BarIndex()==0;
Sell=1;

// these two new options can be set on per-symbol basis
// how many CALENDAR DAYS
// an early exit (redemption) fee is applied
SetOption("EarlyExitDays", 180 );
// early redemption fee (in percent)
SetOption("EarlyExitFee", 2 );

```

(note 180 calendar days is 128 or 129 trading days)

```

// how to set it up on per-symbol basis?
// it is simple - use 'if' statement
if( Name() == "SYMBOL1" )
{
    SetOption("EarlyExitDays", 180 );
    SetOption("EarlyExitFee", 2 );
}

if( Name() == "SYMBOL2" )
{
    SetOption("EarlyExitDays", 30 );
    SetOption("EarlyExitFee", 1 );
}

```

#### See Also:

[Backtesting your trading ideas](#) article.

[Backtesting systems for futures contracts](#) article.

[Using AFL editor](#) section of the guide.

[Insider guide to backtester](#) (newsletter 1/2002)

## Reading backtest report

To view the report of last backtest simply click **Report** button in the automatic analysis window. To view results of ALL past backtest, click drop down arrow on the **Report** button and choose **Report Explorer** option. This will display the Report Explorer window that will show the list of all backtests performed. If you double click on the line – detailed report will be shown.

New report is hugely enhanced compared to old one. It includes separate statistics for all, long and short sides as well as large number of new metrics. You can get short help on given figure by hovering your mouse over given field name. You will see the description in the tooltip. Short explanations are provided also below:

**Exposure %** – 'Market exposure of the trading system calculated on bar by bar basis. Sum of bar exposures divided by number of bars. Single bar exposure is the value of open positions divided by portfolio equity.

**Net Risk Adjusted Return %** – Net profit % divided by Exposure %

**Annual Return %** – Compounded Annual Return % (CAR)

**Risk Adjusted Return %** – Annual return % divided by Exposure %

**Avg. Profit/Loss** – (Profit of winners + Loss of losers)/(number of trades)

**Avg. Profit/Loss %** – '(% Profit of winners + % Loss of losers)/(number of trades)

**Avg. Bars Held** – sum of bars in trades / number of trades

**Max. trade drawdown** – The largest peak to valley decline experienced in any single trade. The lower the better

**Max. trade % drawdown** – The largest peak to valley percentage decline experienced in any single trade. The lower the better

**Max. system drawdown** – The largest peak to valley decline experienced in portfolio equity. The lower the better

**Max. system % drawdown** – The largest peak to valley percentage decline experienced in portfolio equity. The lower the better

**Recovery Factor** – Net profit divided by Max. system drawdown

**CAR/MaxDD** – Compound Annual % Return divided by Max. system % drawdown. Good if bigger than 2

**RAR/MaxDD** – Risk Adjusted Return divided by Max. system % drawdown. Good if bigger than 2.

**Profit Factor** – Profit of winners divided by loss of losers

**Payoff Ratio** – Ratio average win / average loss

**Standard Error** – Standard error measures chopiness of equity line. The lower the better.

**Risk–Reward Ratio** – Measure of the relation between the risk inherent in a trading the system compared to its potential gain. Higher is better. Calculated as slope of equity line (expected annual return) divided by its standard error.

**Ulcer Index** – Square root of sum of squared drawdowns divided by number of bars

**Ulcer Performance Index** – (Annual profit – Treasury notes profit)/Ulcer Index'>Ulcer Performance Index. Currently treasury notes profit is hardcoded at 5.4. In future version there will be user–setting for this.

**Sharpe Ratio of trades** – Measure of risk adjusted return of investment. Above 1.0 is good, more than 2.0 is very good. More information <http://www.stanford.edu/~wfs Sharpe/art/sr/sr.htm> . Calculation: first average percentage return and standard deviation of returns is calculated. Then these two figures are annualized by multiplying them by ratio (NumberOfBarsPerYear)/(AvgNumberOfBarsPerTrade). Then the risk free rate of return is subtracted (currently hard–coded 5) from annualized average return and then divided by annualized standard deviation of returns.

**K–Ratio** – Detects inconsistency in returns. Should be 1.0 or more. The higher K ratio is the more consistent return you may expect from the system. Linear regression slope of equity line multiplied by square root of sum of squared deviations of bar number divided by standard error of equity line multiplied by square root of number of bars. More information: Stocks & Commodities V14:3 (115–118): Measuring System Performance by Lars N. Kestner

#### See Also:

[Old backtest report](#)

[Backtesting your trading ideas](#) article.

[Portfolio Backtesting](#) article.

[Backtesting systems for futures contracts](#) article.

[Using AFL editor](#) section of the guide.

[Insider guide to backtester](#) (newsletter 1/2002)

## How to optimize trading system

*NOTE: This is fairly advanced topic. Please read previous AFL tutorials first.*

### Introduction

AmiBroker 3.70 brings a new feature to Automatic Analysis window called "Optimization".

The idea behind an optimization is simple. First you have to have a trading system, this may be a simple moving average crossover for example. In almost every system there are some parameters (as averaging period) that decide how given system behaves (i.e. is it well suited for long term or short term, how does it

react on highly volatile stocks, etc). The optimization is the process of finding optimal values of those parameters (giving highest profit from the system) for a given symbol (or a portfolio of symbols). AmiBroker is one of the very few programs that allow you to optimize your system on multiple symbols at once.

To optimize your system you have to define from one upto ten parameters to be optimized. You decide what is a minimum and maximum allowable value of the parameter and in what increments this value should be updated. AmiBroker then performs multiple back tests the system using ALL possible combinations of parameters values. When this process is finished AmiBroker displays the list of results sorted by net profit. You are able to see the values of optimization parameters that give the best result.

### Writing AFL formula

Optimization in back tester is supported via new function called optimize. The syntax of this function is as follows:

```
variable = optimize( "Description", default, min, max, step );
```

where:

variable – is normal AFL variable that gets assigned the value returned by optimize function. With normal backtesting, scanning, exploration and commentary modes the optimize function returns *default* value, so the above function call is equivalent to: `variable = default;`

In optimization mode optimize function returns successive values from *min* to *max* (inclusively) with *step* stepping.

"Description" is a string that is used to identify the optimization variable and is displayed as a column name in the optimization result list.

*default* is a default value that optimize function returns in exploration, indicator, commentary, scan and normal back test modes

*min* is a minimum value of the variable being optimized

*max* is a maximum value of the variable being optimized

*step* is an interval used for increasing the value from *min* to *max*

Notes:

- AmiBroker supports upto 10 calls to optimize function (therefore upto 10 optimization variables)
- Each call to optimize generate  $(max - min)/step$  optimization loops and multiple calls to optimize multiply the number of runs needed. For example optimizing two parameters using 10 steps will require  $10 \times 10 = 100$  optimization loops.
- Call optimize function only ONCE per variable at the beginning of your formula as each call generates a new optimization loops
- Multiple-symbol optimization is fully supported by AmiBroker

### Examples

1. Single variable optimization:



```
sigavg = Optimize( "Signal average", 9, 2, 20, 1 );

Buy = Cross( MACD( 12, 26 ), Signal( 12, 26, sigavg ) );
Sell = Cross( Signal( 12, 26, sigavg ), MACD( 12, 26 ) );
```

## 2. Two-variable optimization (suitable for 3D charting)

```
per = Optimize( "per", 2, 5, 50, 1 );
Level = Optimize( "level", 2, 2, 150, 4 );

Buy=Cross( CCI(per), -Level );
Sell = Cross( Level, CCI(per) );
```

## 3. Multiple (3) variable optimization:

```
mfast = Optimize( "MACD Fast", 12, 8, 16, 1 );
mslow = Optimize( "MACD Slow", 26, 17, 30, 1 );
sigavg = Optimize( "Signal average", 9, 2, 20, 1 );

Buy = Cross( MACD( mfast, mslow ), Signal( mfast, mslow, sigavg ) );
Sell = Cross( Signal( mfast, mslow, sigavg ), MACD( mfast, mslow ) );
```

After entering the formula just click on **Optimize** button in "Automatic Analysis" window. AmiBroker will start testing all possible combinations of optimization variables and report the results in the list. After optimization is done the list of result is presented sorted by the Net % profit. As you can sort the results by any column in the result list it is easy to get the optimal values of parameters for the lowest drawdown, lowest number of trades, largest profit factor, lowest market exposure and highest risk adjusted annual % return. The last columns of result list present the values of optimization variables for given test.

When you decide which combination of parameters suits your needs the best all you need to do is to replace the default values in optimize function calls with the optimal values. At current stage you need to type them by hand in the formula edit window (the second parameter of optimize function call).

## Displaying 3D animated optimization charts

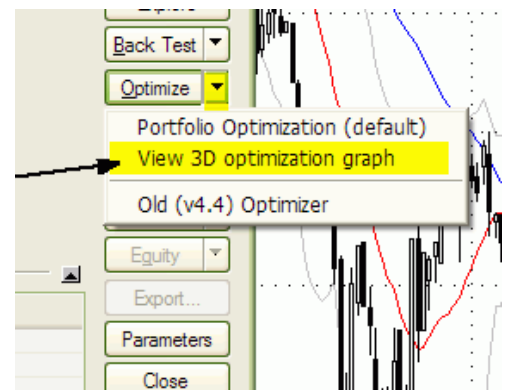
To display 3D optimization chart, you need to run two-variable optimization first. Two variable optimization needs a formula that has 2 Optimize() function calls. An example two-variable optimization formula looks like this:

```
per = Optimize( "per", 2, 5, 50, 1 );
Level = Optimize( "level", 2, 2, 150, 4 );

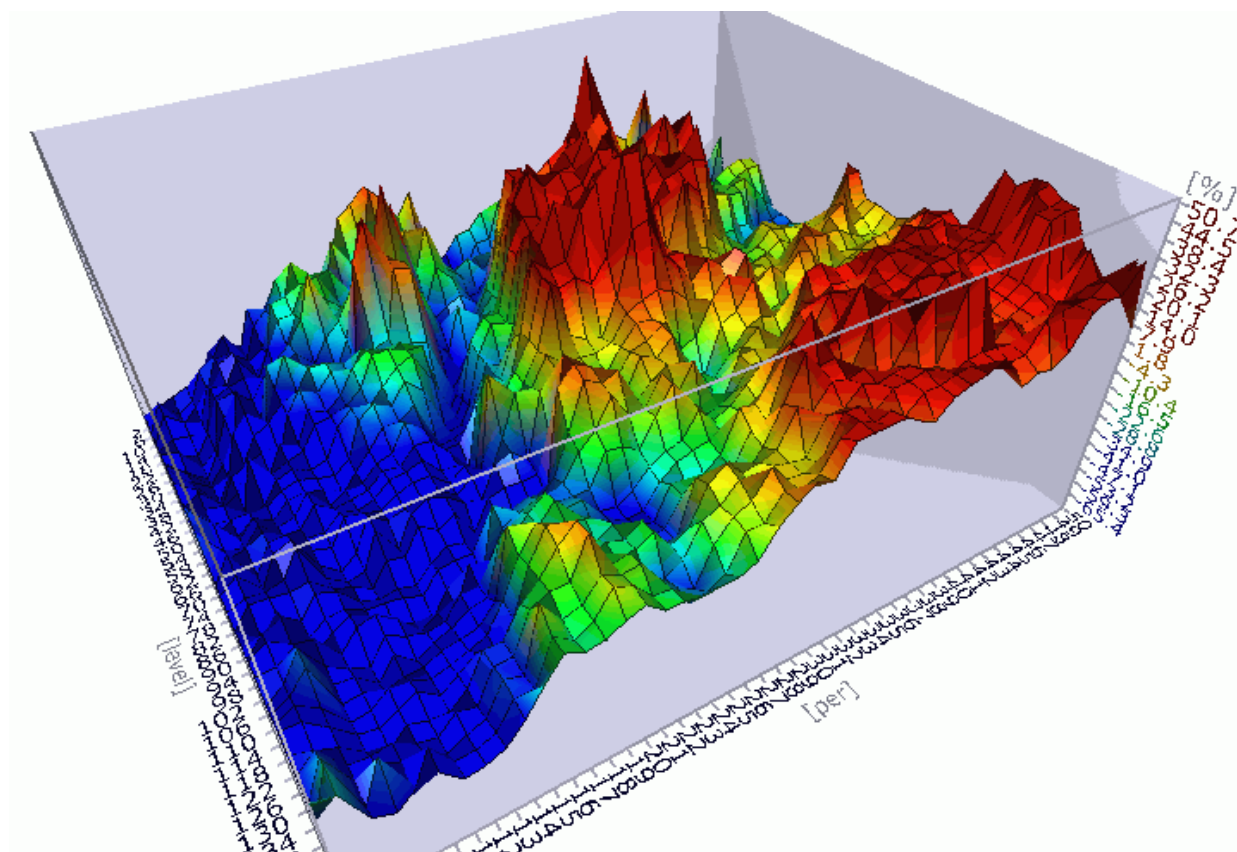
Buy=Cross( CCI(per), -Level );
Sell = Cross( Level, CCI(per) );
```

After entering the formula you need to click "Optimize" button.

Once optimization is complete you should click on the drop down arrow on **Optimize** button and choose **View 3D optimization graph**. In a few seconds a colorful three-dimensional surface plot will appear in a 3D chart viewer window. An example 3D chart



generated using above formula is shown below.



By default the 3D charts display values of Net profit against optimization variables. You can however plot 3D surface chart for any column in the optimization result table. Just click on the column header to sort it (blue arrow will appear indicating that optimization results are sorted by selected column) and then choose **View 3D optimization graph** again.

By visualizing how your system's parameters affect trading performance, you can more readily decide which parameter values produce "fragile" and which produce "robust" system performance. Robust settings are regions in the 3D graph that show gradual rather than abrupt changes in the surface plot. 3D optimization charts are great tool to prevent curve-fitting. Curve-fitting (or over-optimization) occurs when the system is more complex than it needs to be, and all that complexity was focused on market conditions that may never happen again. Radical changes (or spikes) in the 3D optimization charts show clearly over-optimization areas. You should choose parameter region that produces a broad and wide plateau on 3D chart for your real life trading. Parameter sets producing profit spikes will not work reliably in real trading.

### 3D chart viewer controls

AmiBroker's 3D chart viewer offers total viewing capabilities with full graph rotation and animation. Now you can view your system results from every conceivable perspective. You can control the position and other parameters of the chart using the mouse, toolbar and keyboard shortcuts, whatever you find easier for you. Below you will find the list.

*Mouse controls:*

- to Rotate – hold down LEFT mouse button and move in X/Y directions
- to Zoom-in, zoom-out – hold down RIGHT mouse button and move in X/Y directions
- to Move (translate) – hold down LEFT mouse button and CTRL key and move in X/Y directions
- to Animate – hold down LEFT mouse button, drag quickly and release button while dragging

#### *Keyboard controls:*

SPACE – animate (auto-rotate)  
 LEFT ARROW KEY – rotate vert. left  
 RIGHT ARROW KEY – rotate vert. right  
 UP ARROW KEY – rotate horiz. up  
 DOWN ARROW KEY – rotate horiz. down  
 NUMPAD + (PLUS) – Near (zoom in)  
 NUMPAD – (MINUS) – Far (zoom out)  
 NUMPAD 4 – move left  
 NUMPAD 6 – move right  
 NUMPAD 8 – move up  
 NUMPAD 2 – move down  
 PAGE UP – water level up  
 PAGE DOWN – water level down

## **Back-testing systems for futures contracts**

### **Introduction**

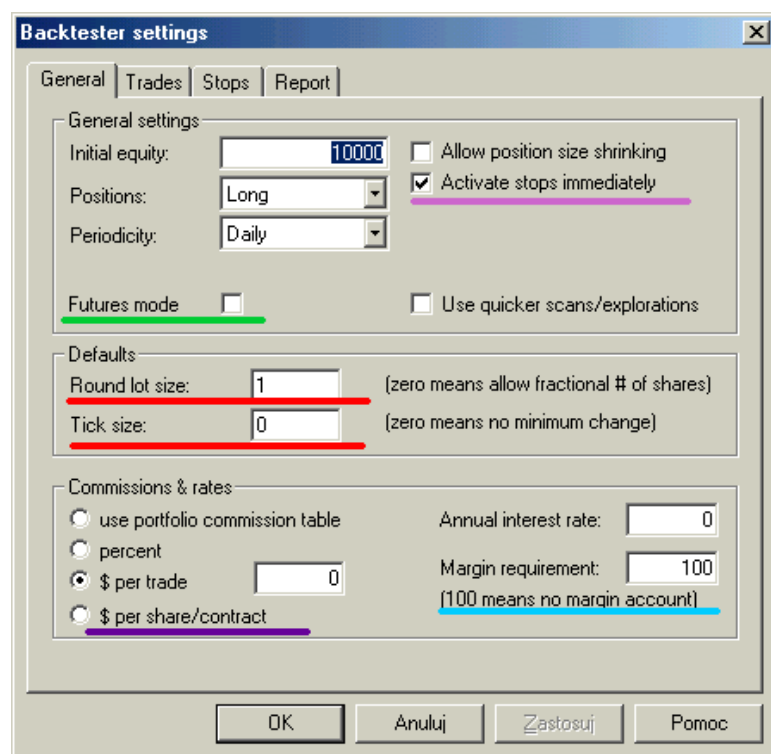
Before you read this article you should read first "[Backtesting your trading ideas](#)" section as it gives necessary background of backtesting in general.

When you open long position on stocks you just buy given number of shares at given price, then after some time you sell them and your profit is given by difference between sell and buy price multiplied by number of shares. If you want to open long position on future contract you pay a deposit – margin – for each contract. The margin is just a little part of full contract value (for example 10%). So you can buy 10 contracts paying no more than full value of one contract. This gives you a leverage that makes trading futures more risky than trading stocks. When price of the contract changes your profit/loss changes accordingly. If contract's point value is 1 each 1\$ change in contract price represents 1\$ profit/loss per contract – like in stocks. But futures can have point value different than 1. If, for example, point value is 5 each 1 point change in price of the contract represents 5\$ profit/loss in your equity. When you close position you get the margin deposit back, so your profit/loss is given by number of contracts multiplied by point value multiplied by difference between sell and buy prices.

### **Futures mode of the backtester**

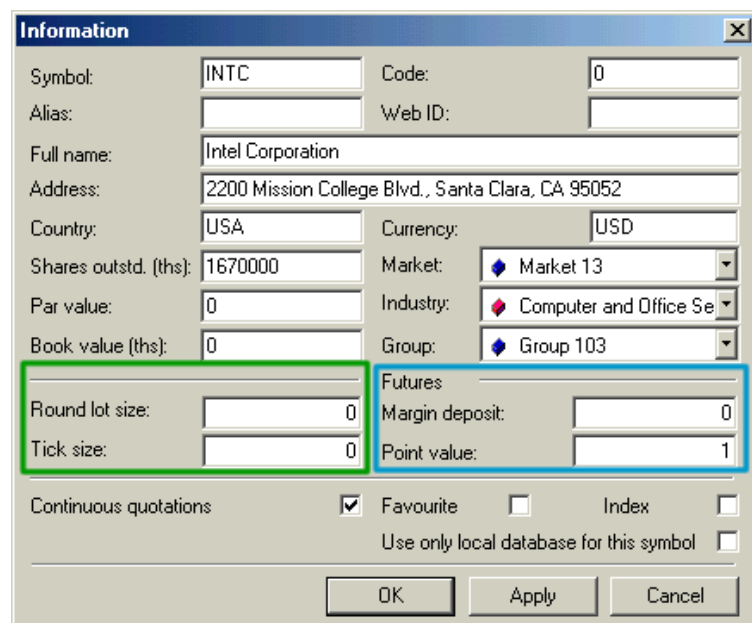
There are 3 futures-only settings in the backtester:

- Futures mode check box (Settings-General page)
- Margin deposit (Symbol-Information page)
- Point value (Symbol-Information page)



Futures mode check box in the settings page (underscored with green line in the picture above) is the key to backtesting futures. It instructs backtester to use margin deposit and point value in calculations.

The remaining settings are per-symbol and they are accessible from Symbol->Information window.



### Margin deposit

The margin is the amount of money required to open single contract position. You can specify per-symbol margin in the Symbol-Information page (picture above). Positive values describe margin value in dollars, while negative express margin value as percentage of contract price. Margin value of zero is used for stocks

(no margin). Margin can be also specified in the formula by using MarginDeposit reserved variable:

```
MarginDeposit = 675;
```

In the Futures mode margin setting is used to determine how many contracts can be purchased. Let's suppose that your initial equity is set to \$50000 and you want to invest upto 20% of equity in single trade and the margin deposit is \$675. In that case your "desired" position size is  $50'000 * 0.2 = 10'000$ . Provided that you have set round lot size to 1, the backtester will "buy"  $10000/675 = (\text{integer})14.8148 = 14$  contracts, and true position value will be \$9450 (18.9% of the initial equity).

To simulate this in AmiBroker you would need to enter 50000 in the Initial Equity field in the backtester, switch on futures mode, and setup remaining parameters in your formula:

```
PositionSize = -20; // use 20% of equity
MarginDeposit = 675; // this you can set also in the
Symbol-Information page
RoundLotSize = 1; // this you can set also in the Settings page
```

All further trades will use the same logic but position will be sized according to current cumulated equity instead of initial equity level, unless you specify fixed position size in your formula ( PositionSize = 10000 for example).

### *Point value*

Point-value is per-symbol setting (definable in Symbol-Information window – (picture above)) that determines the amount of profit generated by one contract for a one point increase in price. Example: copper is quoted in cents per pound, a price quote of 84.65 (or 8465) equals 84 cents and 65/100 of a cent per pound. A change of +.37 or 37 represents 37/100ths of a cent you will normally hear it quoted as 37 points. But because of the fact that point value for copper is 2.5 every point change gives \$2.5 profit/loss, so in this example profit/loss for the day would be  $2.5 * 37 = \$92.50$ .

You can also set it from the formula level using PointValue reserved variable, for example:

```
PointValue = 2.5;
```

Note: When you load old database AmiBroker presets point value field to 1 and assumes that by default 1 point represents one dollar so one dollar change gives one dollar profit/loss. This is done to ensure that you get correct results even if you (by mistake) run futures mode test on stocks.

Note 2: Although point value setting affects (multiplies) profits/losses it does NOT affect built-in stops. The stops ALWAYS operate on price movement alone. So you should be aware that setting 10% profit target stop will result in 25% profit on trade exited by this stop when point value is set to 2.5.

## **Simple cases**

### *Points-only test*

Points only test is equivalent to trading just one contract. This can be easily accomplished using Futures mode of the backtester and adding the following one line to your formula:

```
PositionSize = MarginDeposit = 1;
```

*Trading 'n' contracts*

In a similar way you can setup your formula so it always trades say 7 contracts. All you need to do is to add the following to your formula:

```
NumContracts = 7;
PositionSize = NumContracts * MarginDeposit;
```

## Pyramiding (scaling in/out) and mutiple currencies in the portfolio backtester

**IMPORTANT:** Please read first Tutorial: [Backtesting your trading ideas article](#) and [Portfolio Backtesting](#)

Starting from version 4.70 portfolio backtester allows position scaling and supports multiple currencies. Note that these advanced features are supported by PORTFOLIO backtester only. Old single–security backtester and single–security equity() function do NOT support these features.

### Pyramiding / Scaling

Two special constants: sigScaleIn / sigScaleOut added to provide means to tell the backtester when you want to scale–in/out

All you have to do to implement pyramiding is to:

- Assign sigScaleIn to BUY/SHORT variable if you want to scale–in (increase size of) LONG/SHORT position
- Assign sigScaleOut to BUY/SHORT variable if you want to scale–out (decrease size of) LONG/SHORT position

Scaling size is defined by PositionSize variable which in case of scaling defines not absolute positionsize but dollar increase or decrease.

**IMPORTANT:** Please note that backtester treats trade that you scale–in/out as SINGLE trade (i.e. will show single row in trade list). The only difference versus plain trade is that it will calculate average entry price (and avg. entry fx rate) based on all partial entries and average exit price (and avg. exit fx rate) based on all parial exits and will show average prices in entry/exit price field. The commission is of course applied correctly to each (partial) entry/exit depending on partial buy/sell size.

If you want to see details about scaling you have to run backtest in "DETAIL LOG" mode as only then you will see how scaling–in /out works and how average prices are calculated.

Note also that scaling–in/–out and multiple–currency support is available only in portfolio backtester. Old backtester as well as Equity() function do NOT handle scaling–in/out nor multiple currencies (they simply ignore scaling commands).

Easy examples:

**Example 1: dollar–cost averaging (each month you buy stocks for fixed dollar amount)**

```

FixedDollarAmount = 500;
MonthBegin = Month() != Ref( Month(), -1 );

FirstPurchase = Cum( MonthBegin ) == 1;

Buy = IIf( FirstPurchase, 1, // True (or 1) represents regular buy signal
          IIf( MonthBegin, sigScaleIn, // each month increase position
              0 ) ); // otherwise no signal

Sell = 0; // we do not sell
PositionSize = FixedDollarAmount;

```

**Example 2: dollar-cost averaging**  
*(simplified formula because AB treats first sigScaleIn as buy anyway)*

```

FixedDollarAmount = 500;
MonthBegin = Month() != Ref( Month(), -1 );

FirstPurchase = Cum( MonthBegin ) == 1;

Buy = IIf( MonthBegin, sigScaleIn, 0 ); // each month increase position

Sell = 0; // we do not sell

PositionSize = FixedDollarAmount;

```

**Example 3: increasing position when profit generated by trade without pyramiding becomes greater than 5% and decreasing position when loss is greater than -5%**

```

// percent equity change threshold when pyramiding is performed
PyramidThreshold = 5;

// regular trading rules (no pyramiding)
Buy = Cross( MACD(), Signal() );
Sell = Cross( Signal(), MACD() );

e = Equity(1); // generate equity without pyramiding effect

PcntProfit = 100 * ( e - ValueWhen( Buy, e ) ) / ValueWhen( Buy, e );

InTrade = Flip( Buy, Sell );

// ExRem is used here to ensure that scaling-in/out occurs
// only once since trade entry
DoScaleIn = ExRem( InTrade AND PcntProfit > PyramidThreshold, Sell );
DoScaleOut = ExRem( InTrade AND PcntProfit < -PyramidThreshold, Sell );

// modify rules to handle pyramiding

```

```
Buy = Buy + sigScaleIn * DoScaleIn + sigScaleOut * DoScaleOut;
```

```
PositionSize = IIf( DoScaleOut, 500, 1000 ); // enter and scale-in size $1000,
scale-out size: $500
```

#### Example 4: partial exit (scaling out) on profit target stops

Example of code that exits 50% on first profit target, 50% on next profit target and everything at trailing stop:

```
Buy = Cross( MA( C, 10 ), MA( C, 50 ) );
Sell = 0;

// the system will exit
// 50% of position if FIRST PROFIT TARGET stop is hit
// 50% of position is SECOND PROFIT TARGET stop is hit
// 100% of position if TRAILING STOP is hit

FirstProfitTarget = 10; // profit
SecondProfitTarget = 20; // in percent
TrailingStop = 10; // also in percent

priceatbuy=0;
highsincebuy = 0;

exit = 0;

for( i = 0; i < BarCount; i++ )
{
    if( priceatbuy == 0 AND Buy[ i ] )
    {
        priceatbuy = BuyPrice[ i ];
    }

    if( priceatbuy > 0 )
    {
        highsincebuy = Max( High[ i ], highsincebuy );

        if( exit == 0 AND
            High[ i ] >= ( 1 + FirstProfitTarget * 0.01 ) * priceatbuy )
        {
            // first profit target hit - scale-out
            exit = 1;
            Buy[ i ] = sigScaleOut;
        }

        if( exit == 1 AND
            High[ i ] >= ( 1 + SecondProfitTarget * 0.01 ) * priceatbuy )
        {
            // second profit target hit - exit
            exit = 2;
        }
    }
}
```



```

        SellPrice[ i ] = Max( Open[ i ], ( 1 + SecondProfitTarget * 0.01 ) *
priceatbuy );
    }

    if( Low[ i ] <= ( 1 - TrailingStop * 0.01 ) * highsincebuy )
    {
        // trailing stop hit - exit
        exit = 3;
        SellPrice[ i ] = Min( Open[ i ], ( 1 - TrailingStop * 0.01 ) *
highsincebuy );
    }

    if( exit >= 2 )
    {
        Buy[ i ] = 0;
        Sell[ i ] = exit + 1; // mark appropriate exit code
        exit = 0;
        priceatbuy = 0; // reset price
        highsincebuy = 0;
    }
}

SetPositionSize( 50, spsPercentOfEquity );
SetPositionSize( 50, spsPercentOfPosition * ( Buy == sigScaleOut ) ); // scale
out 50% of position

```

## Multitple Currency Support

The portfolio backtester allows to backtest systems on securities denominated in different currencies. It includes ability to use historical (variable) currency rates. Currency rates are definable in "Currencies" page in the preferences. The currency in which given symbol is denominated in can be entered in Symbol->Information page.

"Currencies" page in Preferences – allows to define base currency and exchange rates (fixed or dynamic) for different currencies. This allows to get correct backtest results when testing securities denominated in different currency than your base portfolio currency.

How does AB know whether I want the fixed or dynamic quote?

There are following requirements to use currency adjustments:

- Symbol->Information, "Currency" field shows currency different than BASE currency
- Appropriate currency (defined in Symbol) has matching entry in Preferences->Currencies page
- the dynamic rate "FX SYMBOL" defined in the preferences EXISTS in your database and HAS QUOTES for each day under analysis range.

What is "INVERSE" check box for in the preferences?

Let's for example take EURUSD.

When "USD" is your BASE currency then EUR exchange rate would be "straight" EURUSD fx (i.e. 1.3).  
But when "EUR" is your BASE currency then USD exchange rate would be INVERSE of EURUSD (i.e. 1/1.3).  
Opposite would be true with FX rates like USDJPY (which are already "inverse").

## Using formula-based alerts

### Introduction

AmiBroker allows you to define formula-based alerts. When alert is triggered a text can be displayed, user-defined sound played back, e-mail notification can be sent and any external application can be launched. This is all handled by single AlertIF function.

By default all alerts generate text that is displayed in the Alert Output window.

To show this window you have to select View->Alert Output menu.

There is also [Easy Alerts](#) window that allows you to define simple alerts that do not require any coding (but do not offer full flexibility of AlertIF function).

### Settings

Alert – related settings are present in the "Alerts" tab of Tools->Preferences window.

It allows to define e-mail account settings, test sound output and define which parts of AmiBroker can generate alerts via AlertIF function.

E-mail setting page now allows to choose among most popular authorization schemes like: AUTH LOGIN (most popular), POP3-before-SMPT (popular), CRAM-MD5, LOGIN PLAIN.

"Enable alerts from" checkboxes allow you to selectively enable/disable alerts generated by Automatic analysis, Commentary/Interpretation and custom indicators.

Alert output window now has an additional column that shows the source of alert – if this is Automatic Analysis, Commentary or one of your custom indicators. This makes it easier to find out which part of AmiBroker generates alerts.

### AlertIF function

AlertIF function is similar to WriteIF. But instead of just writing the text to the output window (commentary/interpretation) it allows to:

- direct the customized text to "alert output" window,
- make a sound (just by computer beeper or from .WAV file)
- send an e-mail
- launch any external application

The syntax is as follows:

```
AlertIf( BOOLEAN_EXPRESSION, command, text, type = 0, flags = 1+2+4+8, lookback = 1 );
```

1. *BOOLEAN\_EXPRESSION* is the expression that if evaluates to True (non zero value) triggers the alert. If it evaluates to False (zero value) no alert is triggered. Please note that only *lookback* most recent bars are considered.

2. The *command* string defines the action taken when alert is triggered. If it is empty the alert *text* is simply

displayed in the Alert output window (View→Alert Output). Other supported values of *command* string are:

SOUND *the-path-to-the-WAV-file*  
 EMAIL  
 EXEC *the-path-to-the-file-or-URL* <optional args>

SOUND command plays the WAV file once.

EMAIL command sends the e-mail to the account defined in the settings (Tools→Preferences→E-mail). The format of the e-mail is as follows:

Subject: Alert type\_name (*type*) Ticker on Date/Time  
 Body: *text*

EXEC command launches external application or file or URL specified after EXEC command. <optional args> are attached after file name and *text* is attached at the end

3. *Text* defines the text that will be printed in the output window or sent via e-mail or added as argument to the application specified by EXEC command

4. *Type* defines type of the alert. Pre-defined types are 0 – default, 1 – buy, 2 – sell, 3 – short, 4 – cover. YOU may specify higher values and they will get name "other"

5. *Flags* control behaviour of AlertIf function. This field is a combination (sum) of the following values: ( 1 – display text in the output window, 2 – make a beep (via computer speaker), 4 – don't display repeated alerts having the same type, 8 – don't display repeated alerts having the same date/time) By default all these options are turned ON.

6. *lookback* parameter controls how many recent bars are checked

Examples:

```
Buy = Cross( MACD(), Signal() );
Sell = Cross( Signal(), MACD() );
Short = Sell;
Cover = Buy;

AlertIf( Buy, "EMAIL", "A sample alert on "+FullName(), 1 );

AlertIf( Sell, "SOUND C:\\Windows\\Media\\Ding.wav", "Audio alert", 2 );

AlertIf( Short, "EXEC Calc.exe", "Launching external application", 3 );

AlertIf( Cover, "", "Simple text alert", 4 );
```

Note EXEC command uses ShellExecute function and allows not only EXE files but URLs too.

## Notes

1. Please note that by default AlertIf function does not generate repetitive signals when the same scan is run multiple times. During experimentation you may prefer to get repeated signals in subsequent scans. To do so you should change default flags to 1 + 2:

```
AlertIF( condition, "", "Text", 1, 1+2 );
```

2. If you want to generate the alert only on COMPLETED bar you may need to add this code:

```
barcomplete = BarIndex() < LastValue(BarIndex());

AlertIF( barcomplete AND condition, "", "Text", 1 );
```

## Using interpretation window

Note: Please read [How to write your own chart commentary](#) article before proceeding.

Interpretation window (View->Interpretation) shows chart-sensitive commentaries. To add a interpretation just use [Formula Editor](#) and add commentary code after the code for the indicator. Please note that to get the best performance you should use conditional statement that ensures that interpretation code is executed only in "commentary" mode.

```
if( Status("action") == actionCommentary )
{
// printf statements here....
}
```

Example:

```
Plot( Close, "Price", -1, 64 );
Plot( SAR( Prefs( 50 ), Prefs( 51 ) ), "SAR", -17, 8+16 );

if( Status("action") == actionCommentary )
{
printf("The Parabolic SAR provides excellent exit points. \n");
printf("You should Close long positions when the price falls below\n");
printf("the SAR AND Close Short positions when the price rises above the SAR.\n");
printf( WriteIf( Graph1 > Close, "SAR is above close", "SAR is below close" ) );
}
```

## Multiple Time Frame support in AFL

Release 4.41 brings ability to use multiple time frames (bar intervals) in single formula. The time frame functions can be divided into 3 functional groups:

1. switching time frame of build-in O, H, L, C, V, OI, Avg arrays: **TimeFrameSet**, **TimeFrameRestore**
2. compressing/expanding single arrays to/from specified interval: **TimeFrameCompress**, **TimeFrameExpand**
3. immediate access to price/volume arrays in different time frame: **TimeFrameGetPrice**

**First group** is used when your formula needs to perform some calculations on indicators in different time frame than currently selected one. For example if you need to calculate 13-bar moving average on 5 minute data and 9 bar exponential average from hourly data while current interval is 1 minute you would write:

```

TimeFrameSet( in5Minute ); // switch to 5 minute frame

/* MA now operates on 5 minute data, ma5_13 holds time-compressed 13 bar MA of
5min bars */

ma5_13 = MA( C, 13 );

TimeFrameRestore(); // restore time frame to original

TimeFrameSet( inHourly ); // switch now to hourly

mah_9 = EMA( C, 9 ); // 9 bar moving average from hourly data

TimeFrameRestore(); // restore time frame to original

Plot( Close, "Price", colorWhite, styleCandle );

// plot expanded average

Plot( TimeFrameExpand( ma5_13, in5Minute), "13 bar moving average from 5 min
bars", colorRed );
Plot( TimeFrameExpand( mah_9, inHourly), "9 bar moving average from hourly bars",
colorRed );

```

**TimeFrameSet( interval )** – replaces current built-in price/volume arrays: open, high, low, close, volume, openint, avg with time-compressed bars of specified interval once you switched to a different time frame all calculations and built-in indicators operate on selected time frame. To get back to original interval call TimeFrameRestore() function. If you want to call TimeFrameSet again with different interval you have to restore original time frame first using TimeFrameRestore(). Interval is time frame interval in seconds. For example: 60 is one minute bar. You should use convenient constants for common intervals: in1Minute, in5Minute, in15Minute, inHourly, inDaily, inWeekly, inMonthly.

With version 4.70 you can also specify N-tick intervals. This is done by passing NEGATIVE value as interval. For example -5 will give 5-tick bar compression, and -133 will give 133-tick compression. Please note that using N-tick intervals works only if your database uses Tick base time interval set in **File -> Database Settings** dialog.

```
TimeFrameSet( -133 ); // switch to 133-tick interval
```

**TimeFrameRestore()** – restores price arrays replaced by SetTimeFrame. Note that only OHLC, V, OI and Avg built-in variables are restored to original time frame when you call TimeFrameRestore(). All other variables created when being in different time frame remain compressed. To de-compress them to original interval you have to use TimeFrameExpand.

Once you switch the time frame using TimeFrameSet, all AFL functions operate on this time frame until you switch back the time frame to original interval using TimeFrameRestore or set to different interval again using TimeFrameSet. It is good idea to ALWAYS call TimeFrameRestore when you are done with processing in other time frames.

When time frame is switched to other than original interval the results of all functions called since TimeFrameSet are time-compressed too. If you want to display them in original time frame you would need to 'expand' them as described later. Variables created and assigned before call to TimeFrameSet() remain in the

time frame they were created. This behaviour allows mixing unlimited different time frames in single formula.

Please note that you can only compress data from shorter interval to longer interval. So when working with 1-minute data you can compress to 2, 3, 4, 5, 6, ....N-minute data. But when working with 15 minute data you can not get 1-minute data bars. In a similar way if you have only EOD data you can not access intraday time frames.

**Second group:** TimeFrameCompress/TimeFrameExpand allow to compress and expand single arrays to / from different time frames. Especially worth mentioning is TimeFrameExpand that is used to decompress array variables that were created in different time frame. Decompressing is required to properly display the array created in different time frame. For example if you want to display weekly moving average it must be 'expanded' so the data of one weekly bar covers five daily bars (Monday–Friday) of corresponding week.

**TimeFrameExpand**( array, interval, mode = expandLast ) – expands time-compressed array from 'interval' time frame to base time frame ('interval' must match the value used in TimeFrameCompress or TimeFrameSet)

Available modes:

expandLast – the compressed value is expanded starting from last bar within given period (so for example weekly close/high/low is available on Friday's bar)

expandFirst – the compressed value is expanded starting from first bar within given period (so for example weekly open is available from Monday's bar)

expandPoint – the resulting array gets not empty values only for the last bar within given period (all remaining bars are Null (empty)).

Caveat: expandFirst used on price different than open may look into the future. For example if you create weekly HIGH series, expanding it to daily interval using expandFirst will enable you to know on MONDAY what was the high for entire week.

TimeFrameCompress is provided for completeness and it can be used when you want to compress single array without affecting built-in OHLC,V arrays. If you call TimeFrameCompress it does not affect results of other functions.

```
wc = TimeFrameCompress( Close, inWeekly );

/* now the time frame is still unchanged (say daily) and our MA will operate on
daily data */

dailyma = MA( C, 14 );

/* but if we call MA on compressed array, it will give MA from other time frame
*/

weeklyma = MA( wc, 14 ); // note that argument is time-compressed array

Plot( dailyma, "DailyMA", colorRed );

weeklyma = TimeFrameExpand( weeklyma, inWeekly ); // expand for display

Plot( weeklyma, "WeeklyMA", colorBlue );
```

During this formula the time frame remained at original setting we only compressed single array.

**TimeFrameCompress**( array, interval, mode = compressLast )

– compresses single array to given interval using given compression mode available modes:  
 compressLast – last (close) value of the array within interval  
 compressOpen – open value of the array within interval  
 compressHigh – highest value of the array within interval  
 compressLow – lowest value of the array within interval  
 compressVolume – sum of values of the array within interval

```
Graph0 = TimeFrameExpand( TimeFrameCompress( Close, inWeekly, compressLast ),
inWeekly, expandLast );
```

```
Graph1 = TimeFrameExpand( TimeFrameCompress( Open, inWeekly, compressOpen ),
inWeekly, expandFirst );
```

**Third group** consist of just one useful function: TimeFrameGetPrice which allows to reference price and volume from other time frames without switching /compressing/expanding time frames. Just one function call to retrieve price from higher time frame. It allows also to reference not only current but past bars from different time frames.

**TimeFrameGetPrice**( pricefield, interval, shift = 0, mode = expandFirst );

– references OHLCV fields from other time frames. This works immediately without need to call TimeFrameSet at all.

Price field is one of the following: "O", "H", "L", "C", "V", "I" (open interest). Interval is bar interval in seconds. shift allows to reference past (negative values) and future (positive values) data in higher time frame. For example -1 gives previous bar's data (like in Ref function but this works in higher time frame).

Examples:

```
TimeFrameGetPrice( "O", inWeekly, -1 ) // gives you previous week Open price
TimeFrameGetPrice( "C", inWeekly, -3 ) // gives you weekly Close price 3 weeks
ago
TimeFrameGetPrice( "H", inWeekly, -2 ) // gives you weekly High price 2 weeks ago
TimeFrameGetPrice( "O", inWeekly, 0 ) // gives you this week Open price.
TimeFrameGetPrice( "H", inDaily, -1 ) // gives previous Day High when working on
intraday data
```

Shift works as in Ref() function but it is applied to compressed time frame.

Note these functions work like these 3 nested functions

```
TimeFrameExpand( Ref( TimeFrameCompress( array, interval, compress(depending on
field used) ), shift ), interval, expandFirst )
```

therefore if shift = 0 compressed data may look into the future ( weekly high can be known on monday ). If you want to write a trading system using this function please make sure to reference PAST data by using negative shift value.

The only difference is that TimeFrameGetPrice is 2x faster than nested Expand/Compress.

#### **Note on performance of TimeFrame functions:**

a) Measurements done on Athlon 1.46GHz, 18500 daily bars compressed to weekly time frame

TimeFrameGetPrice( "C", inWeekly, 0 ) – 0.0098 sec (9.8 milliseconds)

TimeFrameSet( inWeekly ) – 0.012 sec (12 milliseconds)

TimeFrameRestore( ) – 0.006 sec (6 milliseconds)



TimeFrameCompress( Close, inWeekly, compressLast ); – 0.0097 sec (9.7 milliseconds)  
 TimeFrameExpand( array, inWeekly, expandLast ); – 0.0098 sec (9.8 milliseconds)  
 b) Measurements done on Athlon 1.46GHz, 1000 daily bars compressed to weekly time  
 frameall functions below 0.0007 sec (0.7 millisecond)

## EXAMPLES

EXAMPLE 1: Plotting weekly MACD and cross arrows from daily data

```
TimeFrameSet( inWeekly );
m = MACD(12, 26 ); // MACD from WEEKLY data
TimeFrameRestore();

m1 = TimeFrameExpand( m, inWeekly );

Plot( m1, "Weekly MACD", colorRed );
PlotShapes( Cross( m1, 0 ) * shapeUpArrow, colorGreen );
PlotShapes( Cross( 0, m1 ) * shapeDownArrow, colorGreen );
```

EXAMPLE 2: weekly candlestick chart overlaid on line daily price chart

```
wo = TimeFrameGetPrice( "O", inWeekly, 0, expandPoint );
wh = TimeFrameGetPrice( "H", inWeekly, 0, expandPoint );
wl = TimeFrameGetPrice( "L", inWeekly, 0, expandPoint );
wc = TimeFrameGetPrice( "C", inWeekly, 0, expandPoint );

PlotOHLC( wo, wh, wl, wc, "Weekly Close", colorWhite, styleCandle );
Plot( Close, "Daily Close", colorBlue );
```

EXAMPLE 3: Simplified Triple screen system

```
/* switch to weekly time frame */
TimeFrameSet( inWeekly );
whist = MACD( 12, 26 ) - Signal( 12, 26, 9 );
wtrend = ROC( whist, 1 ); // weekly trend - one week change of weekly macd
histogram
TimeFrameRestore();

/* expand calculated MACD to daily so we can use it with daily signals */
wtrend = TimeFrameExpand( wtrend, inWeekly );

/* elder ray */
bullpower= High - EMA(Close,13);
bearpower= Low - EMA(Close,13);

Buy = wtrend > 0 /* 1st screen: positive weekly trend */
AND
bearpower < 0 AND bearpower > Ref( bearpower, -1 ) /* 2nd screen bear power
negative but rising */
AND
H > Ref( H, -1 ); /* 3rd screen, if prices make a new high */
```

```
BuyPrice = Ref( H, -1 ); // buy stop level;

Sell = 0 ; // exit only by stops
ApplyStop( stopTypeProfit, stopModePercent, 30, True );
ApplyStop( stopTypeTrailing, stopModePercent, 20, True );
```

## Video Tutorials (on-line)

For your convenience we have prepared the following Video Tutorials (in Macromedia Flash format) on our web page:

- [How to install AmiBroker](#)
- [How to use drag-and-drop charting interface](#)
- [How to setup new database with eSignal RT feed \(RT version\)](#)
- [How to setup new database with IQFeed RT feed \(RT version\)](#)
- [How to setup new database with Interactive Brokers \(RT version\)](#)
- [How to use AmiQuote in 'manual' mode](#)
- [How to use chart sheets and layouts](#)
- [How to use layers](#)
- [How to use AFL Code Wizard](#)

For more video tutorials please check:

<http://www.amibroker.com/support.html>



# AmiBroker Reference Guide

- [AmiBroker User Interface Reference](#)
- [ASCII Importer reference](#)
- [AmiBroker's OLE automation object model](#)
- [AmiQuote's OLE automation object model](#)

## Windows

This part describes functionality of AmiBroker windows.

All these windows are asynchronous i.e. you can open as many windows as you like, and work with all of them at the same time.

### Charting

- [Chart window pane](#)
- [Parameters window](#)
- [Study drawing tools](#)
- [Line study properties window](#)
- [Text box properties window](#)
- [Formula editor](#)
- [Risk–yield map window](#)

### Settings

- [Database settings / Intraday settings](#)
- [Preferences](#)
- [Customize tools window](#)

### Symbol / Data

- [Symbol tree](#)
- [Information window](#)
- [Notepad window](#)
- [Quote Editor window](#)
- [Symbol finder window](#)
- [Finance window](#)
- [Profile view](#)
- [Assignment organizer window](#)
- [Composite calculator window](#)
- [Categories window](#)
- [Import Wizard window](#)
- [Metastock importer window](#)
- [Real–time Quote window](#)
- [Easy Alerts window](#)
- [Time/Sales window](#)
- [Bar Replay window](#)

### Analysis/Tools

- [Formula editor](#)
- [Quick review window](#)
- [Analysis window](#)
- [Filter settings window](#)
- [System test settings window](#)
- [Commission schedule window](#)
- [System test report window](#)
- [Commentary window](#)
- [Plugins window](#)
- [Indicator Maintenance wizard](#)

## Chart window pane



This window shows the chart of different technical indicators.

In the bottom of the chart you can see X axis, depending on [Parameter](#) window setting it may or may not display dates, and below you can see scroll bar and chart sheets tab control. Scroll bar can be used to display past quotes, while sheet tab allows to view different chart pages/sheets ([click here to learn more about chart sheets](#)).

To the right you can see Y-axis area (marked with blue color) that shows Y-scale and value labels. Value labels are color fields that display precisely the "last value" of plots. "Last value" is the value of the indicator (or price) for the last currently displayed (rightmost) bar. Y-axis area is used also to move/size chart vertically.

Chart parameters and settings can be adjusted by clicking with RIGHT MOUSE button over chart and choosing **Parameters** option from the [chart context menu](#).

Chart can also be scrolled, resized, moved, shrunk, resized – to learn more about it please read [Tutorial: Basic Charting Guide](#).

## Parameters window

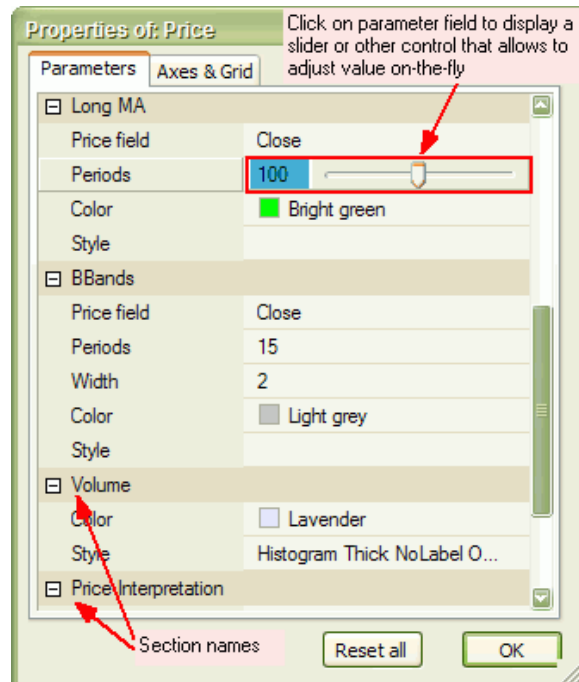
This window allows the user to modify parameters specified in the AFL formula via [Param](#), [ParamStr](#), [ParamColor](#), [ParamStyle](#), [ParamField](#), [ParamToggle](#), [ParamDate](#), [ParamTime](#), [ParamList](#) functions and also to adjust axes and grid settings.

It is accessible via [chart context menu](#) (right click the mouse over the chart pane to see the context menu) : choose **Parameters** and a small window with parameter list will appear. To edit parameter value simply click on the item value field as shown in the picture. Then depending on type of the parameter appropriate control(s) will appear.

For example, if given parameter is a string then text field will appear, and if given parameter is color then color-picker control will allow you to change the color.

When editing numeric parameters you can adjust the value by either entering the value to the edit field or by moving a slider control. To show the edit field – click on the number itself (marked with blue color in the picture below). To show a slider control click next to the number (right-hand side).

If given parameter is a number then slider or the edit field will be shown as in the picture below:



You can move the slider using mouse, <– –> cursor keys and mouse wheel. As changes are made underlying chart is immediately refreshed giving great feedback for the user.

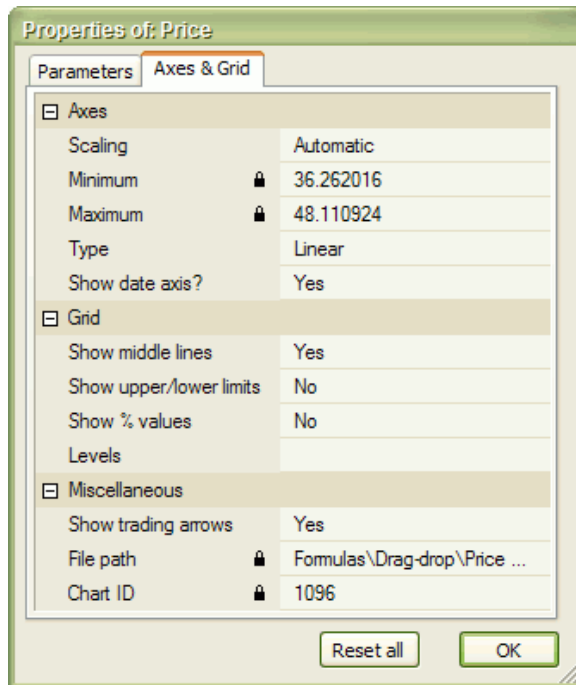
Parameters are grouped into "sections". Sections represent part of the codes surrounded by [\\_SECTION\\_BEGIN/\\_SECTION\\_END](#) markers. To learn more about this check [Tutorial: Using drag-and-drop](#)

interface.

At any time you can press **Reset all** button that will reset all parameters to default values.

For more information on using parameters please read [Tutorial: Using colors, styles, titles and parameters in the indicators](#) and [Tutorial: Using drag-and-drop interface](#).

Parameter window allows also to control axes and grid appearance as well as some other per-chart settings. These controls are available in the second **Axes & Grid** tab as shown below:



The following options are available:

- **Axes**

- ◆ **Scaling:**

- ◇ **Automatic** – minimum and maximum value of Y axis is determined automatically by AmiBroker

- ◇ **Custom** – minimum and maximum value of Y axis are user-defined

- ◆ **Minimum** – minimum Y axis value (this property is locked if automatic scaling is selected, to unlock choose Custom scaling)

- ◆ **Maximum** – maximum Y axis value (this property is locked if automatic scaling is selected, to unlock choose Custom scaling)

- ◆ **Type**

- ◇ **Linear** – use linear Y axis scale

- ◇ **Logarithmic** – use logarithmic Y axis scale

- ◆ **Show date axis** – turn on/off date display on X axis

- **Grid**

- ◆ **Show middle lines** – display automatic Y axis grid lines spaced evenly between minimum and maximum

- ◆ **Show upper/lower limits** – display minimum and maximum Y axis value labels

- ◆ **Show % values** – display values as percents

- ◆ **Levels** – allows to turn on grid lines at some fixed, popular levels such as 30/70, 20/80, 10/90, –100/+100, 0

- **Miscellaneous**

- ◆ **Show trading arrows** – when turned ON this pane will show buy/sell/short/cover arrows generated by corresponding options available from [Automatic Analysis menu](#).
- ◆ **File path** (locked) – shows the path to the formula file that given chart uses
- ◆ **Chart ID** (locked) – shows the numeric value of Chart ID given pane uses. Chart ID does not matter unless you use [Study\(\)](#) function in your formula(s).

## Study drawing tools

AmiBroker's study drawing tools are accessible from **Draw / Fibonacci & Gann** toolbars:



The following tools are available:

- trend line
- ray (new in 4.20)
- extended line (new in 4.20)
- vertical line
- horizontal line
- parallel lines (new in 4.20)
- Regression channels: Raff, standard deviation, standard error (all new in 4.20)
- Fibonacci Retracement study (enhanced in 4.20)
- Fibonacci Time zones study
- Fibonacci Extensions (new in 4.60)
- Fibonacci Time Extensions (new in 4.60)
- Fibonacci Fan
- Fibonacci arc
- Gann Square (new in 4.20)
- Gann Fan (new in 4.20)
- Ellipse tool
- Triangle tool (new in 4.30)
- Andrews' pitchfork (new in 4.30)
- Cycles tool (new in 4.60)
- Arrow tool (new in 4.70)
- Zig-zag tool (new in 4.70)
- Arc tool
- Rectangle
- text box tool

The default *Select* tool (red arrow) is used to select drawing objects and quotations on the chart. If you want to draw given study just switch on appropriate button and start drawing on the chart by pointing the mouse where you want to start the drawing and click-and-hold left mouse button. Then move the mouse. Study tracking line will appear. Release left mouse button when you want to finish drawing. You can also cancel study drawing by pressing ESC (escape) key. For beginners' guide to charting check [Tutorial: Charting guide](#)



**Trend line, Ray, Extended, Vertical, Horizontal**

These tools give different flavours of basic trend line. Trend line gives a line segment, Ray gives right-extended trend line, Extended gives trend line that is extended automatically from both left- and right-sides. Vertical and Horizontal are self-explaining.

**Arrow**

Similar to Trend line but ends with an arrow

**Zig-zag**

Draws a series of connected trend lines. To end drawing press ESC key.

**Parallel**

This tool allows to draw a series of parallel trend line segments. First you draw a trend line as usual, then a second line parallel to the first is automatically created and you can move them around with the mouse. Once you click on the chart it is placed in given position. Then another parallel line appears that can be placed somewhere else. And again, and again. To stop this please either press ESC key or choose "Select" tool.

**Regression channels**

AmiBroker allows to draw easily 3 kinds of regression channels:

- Raff regression channel
- Standard error channel
- Standard deviation channel

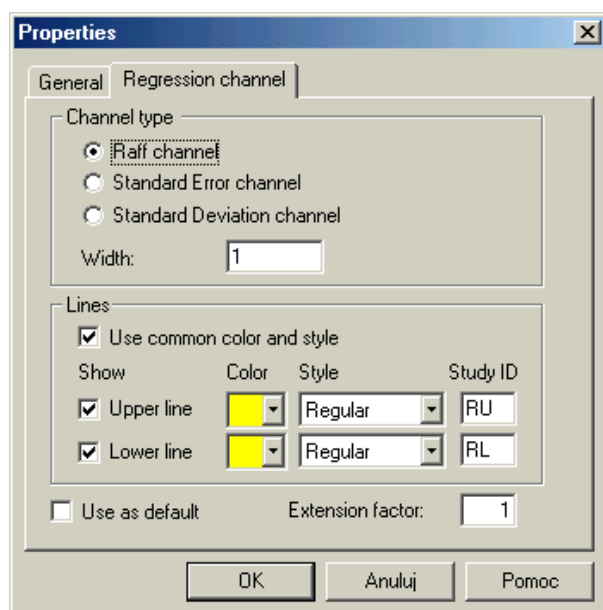
All these channels are based on linear regression trend line.

The Regression Channel is constructed by plotting two parallel, equidistant lines above and below a Linear Regression trendline. The distance between the channel lines to the regression line is the greatest distance that any one high or low price is from the regression line.

Standard Error Channels are constructed by plotting two parallel lines above and below a linear regression trendline. The lines are plotted a specified number of standard errors away from the linear regression trendline.

Standard Deviation Channels are constructed by plotting two parallel lines above and below a linear regression trendline. The lines are plotted a specified number of standard errors away from the linear regression trendline.

You can choose the type of channel by double clicking on the channel study (or choosing **Properties** from right mouse button menu)



If **Use common color and style** box is marked channel lines use the same style and color as regression (middle) line. If it is not marked you can set separate colors and style for upper and lower channel line. You can also switch off completely upper and lower channel lines by unticking **Show Upper line** and **Show Lower line** boxes.

"**Study ID**" column defines study identifier that can be used in your custom formulas to detect crossovers. You can change these IDs if required by simple editing these fields. For more information on Study IDs check [Tutorial: Using studies in AFL formulas](#)  
More information on regression channels is available from [Technical analysis guide](#).

### Ellipse and Arc drawing tools

These new drawing tools are connected to the date/price coordinates (as trend lines) rather than to the screen pixels so they can change the visual shape when displayed at various zoom factors or screen sizes.

To see the properties of these elements you should double-click on the clock-like 3, 6, 9 or 12 hour positions.

### Fibonacci arc

This new drawing tool generates standard Fibonacci-arcs that are controlled by the trend line drawn with a dotted style. To see the properties of the arcs click on the controlling trend line.

Note that arc radius and central point are relative to the controlling trendline and because Fibonacci arcs must be circular regardless of screen size/resolution and zoom factor the position of the arcs may move in date/price domain.

### Fibonacci retracement

First please note that Fibonacci tool works differently depending on the direction of drawing and "show extensions" flag. See the pictures below.

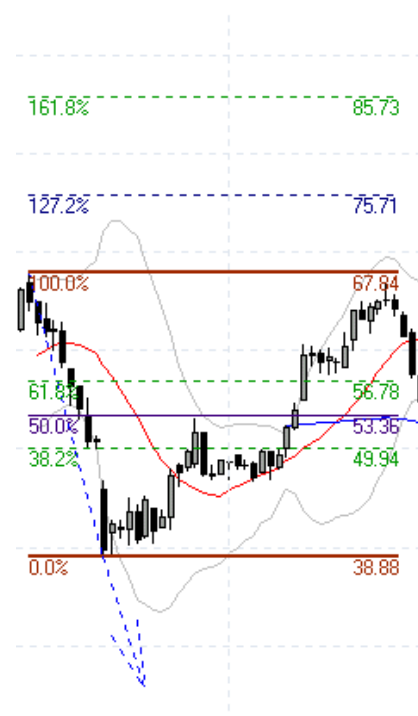


**Upward drawing direction  
Show Extensions ON**



**Upward drawing direction  
Show Extensions OFF**

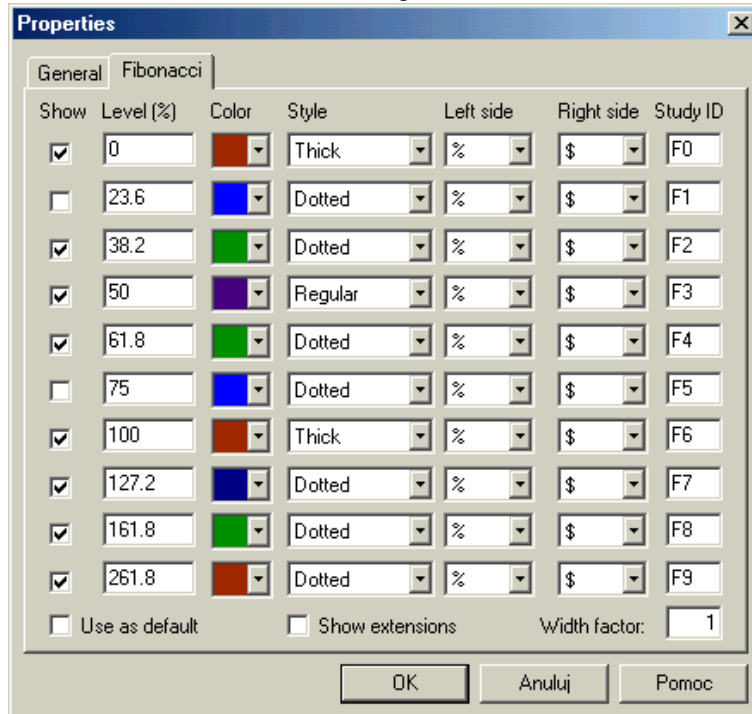
as you can see it shows both retrace levels (38.2, 50, 61.8) and extension levels (127.2, 161.8). If "show extensions" box is OFF the tool shows ONLY retrace levels. It works in a similar way when controlling trend line is drawn downwards.



**Downward drawing direction**  
**Show Extensions ON**

**Downward drawing direction**  
**Show Extensions OFF**

Now more about Fibonacci settings window:



First column "**Show**" switches particular line ON/OFF

Second column "**Level (%)**" defines percentage level. 100 and 0 represent Y-coordinate of begin and end points of controlling trend line.

Third column "**Color**" defines color of the line, Fourth column "**Style**" allows to choose between regular, thick and dotted styles.

Fifth and Sixth columns "**Left side**" and "**Right side**" control display of text that appear on the left and right side of the Fibonacci level line. Empty – means no text, % – means percentage level, \$ – means dollar (point) level.

Seventh column "**Study ID**" defines study identifier that can be used in your custom formulas to detect crossovers. Each Fibonacci level has a separate ID by default F0... F9.

You can change these IDs if required by simple editing these fields.

As described in User's Guide: Tutorial: **Using studies in AFL formulas**

you can easily write the formula that checks for penetration of particular Fibonacci level.

In this example we will detect if the closing price drops F2 (38.2% retracement) level line. The formula is very simple:

```
sell = cross( study( "F2" ), close );
```

Note that study() function accepts two arguments: the first is StudyID two letter code that corresponds to one given in properties dialog; the second argument is chart ID – by default it is 1 (when it is not given at all) and then it references the studies drawn in the main price pane. For checking studies drawn in other panes you should use the codes given above (in the table describing study() function).

Please note that this formula is universal – it will use appropriate level from any symbol that has Fibonacci lines drawn.

This is so because AmiBroker keeps data of all studies drawn in its database.

When you scan using above code – AmiBroker checks if Fibonacci levels are drawn for symbol being currently scanned,

if it finds one – it looks what F2 study is – it finds that this is a fibonacci line 38.2% located (for example for particular symbol) at \$29.06

so AmiBroker internally substitutes study( "F2" ) by \$29.06 (caveat: this is simplification – in fact it internally generates array that represents a trend line) and checks for cross.

**"Extension factor"** decides how far lines are right-extended (in X-axis direction). If you enter 2 you will get lines extended twice as much as default '1'. If you enter 0 Fibonacci level lines will end where controlling trend line ends.

**"Use as default"** – if you check this box and accept the settings by clicking **OK** – all Fibonacci drawings that you will draw later will use these settings.

When using text box tool just type the text in the box, when you want to finish click outside the text box. You can also cancel editing by pressing ESC key.

### ***Fibonacci Extensions***

The Fibonacci Extensions tool is similar to the Fibonacci Retracements tool. The Fibonacci Extensions tool requires a third point. The extensions and retracement levels are drawn from this third point, but based upon the distance between the first two points. A common use of this tool is to first connect two points that represent the endpoints of a major trend (or wave). Then choose the third point to be the endpoint of a retracement of that trend. Extensions are then drawn in the direction of the initial trend, from the third point, using the distance between points one and two as a basis for the extension levels.

The Fibonacci Extensions toolbar button and drawing tool work much like the Andrew's Pitchfork drawing tool. First, click on the Fibonacci Extension button on the toolbar. Then, click three times, once on each of the points that are involved in the Fibonacci Extension. The first click should be on the starting point of the initial trendline. The second click should be on the ending point of the initial trendline. The third click should be on the ending bar of the retracement period.

As with Fibonacci Retracements, there is a great deal of flexibility via Fibonacci settings tab available after clicking on the study with a right mouse and selecting "Properties" from the context menu.

### ***Fibonacci Time Extensions***

Fibonacci Time Extensions tool is used to specify vertical lines at date/time levels which are determined to be probable values of changes in trend based on the market's previous date/time range and a third extension point.

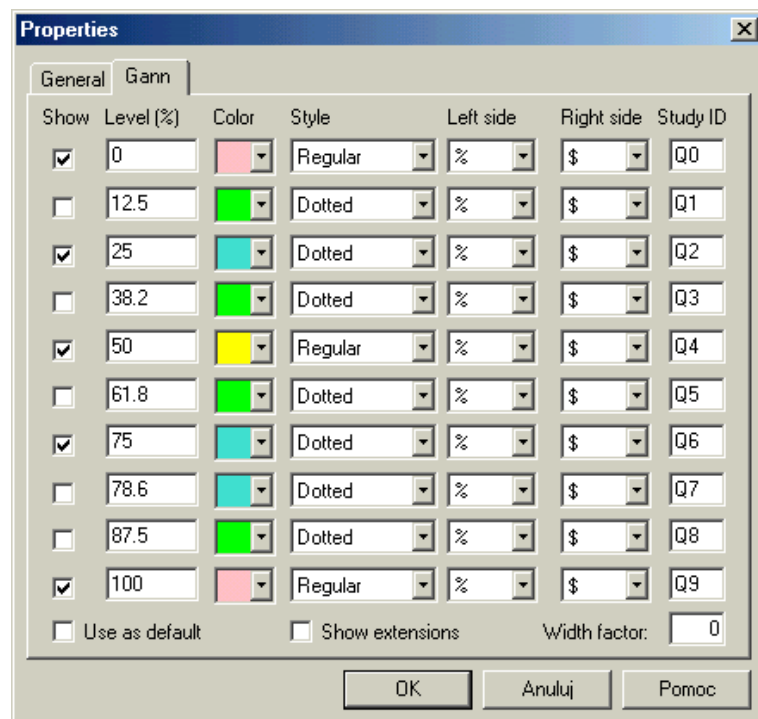
The time extension tool should be used as follows. First, click on the Fibonacci Time extension button on the toolbar. Then select the first range point (typically a major top or bottom of a market) by clicking on the chart where you want the range to begin, then move the mouse pointer to select the second range point by again clicking on the chart where you want the range to end. Extension lines now will be drawn onto the future bars.

As in Fibonacci Price retracement and extensions tools you have complete control over which percentages are used in the Time Extensions tool, and the colours of each of the extension values via Properties dialog.

### **Gann square and Gann Fan**

Gann Squares indicate possible time and price movements from important highs and lows. To draw a Gann Square on a chart move the cursor on the chart to the starting point. The starting point is generally an important High or Low on the chart. Then drag the mouse to the right until a desired ending point is reached. The start and end points will be the corners of the square. The ending point is often to the right of the chart bars. Watch for trends to change directions at the Gann Square levels. As the Gann Square is drawn to the screen the angle of the controlling trend line is shown in the status bar.

### Properties Window



The properties window is used to change the square levels, color, style, thickness, and defaults. Click on any of the Gann Square **Show** entries to add or remove lines. Click in the square **color** box to change the line color. Click on **style** combo boxes to change the line style. Check the **Use as Default** box to save the settings as the default for all subsequent Gann Squares that are drawn. "**Left side**" and "**Right side**" columns control display of text that appear on the left and right side of the Gann lines. Empty – means no text, % – means percentage level, \$ – means dollar (point) level. "**Study ID**" column defines study identifier that can be used in your custom formulas to detect crossovers. You can change these IDs if required by simple editing these fields. For more information on Study IDs check [Tutorial: Using studies in AFL formulas](#)

### Triangle tool

Triangle tool is self-explaining. Drawing a triangle is easy: left-click at the first point, hold down and drag to the second point, then release mouse button and drag to the third point and click once. The controlling triangle will become the pitchfork.

### Andrews' Pitchfork

Andrews pitchfork is a study using parallel trendlines. In constructing the study, starting points are chosen. The first is a major peak or trough on the left side of the chart display. The second and third starting points are chosen to be a major peak and a major trough to the right of the first point. After all starting points have been decided, AmiBroker draws a trendline from the first point (the most left) so that it passes directly between the

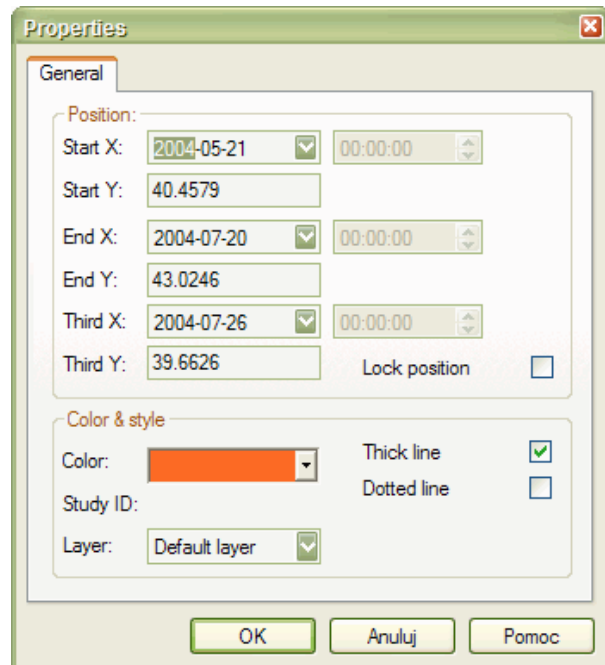
right most points. This line is called the handle of the pitchfork. The second and third trend lines are drawn by AmiBroker beginning at the starting points and parallel to the handle. Dr. Andrews suggested that prices make it to the median line (or handle) about 80% of the time while the price trend is in place. This means that while the basic long term price trend remains intact, Dr. Andrews believed that the smaller trends in price would gravitate toward the median line while the larger price trend remained in tact. When that does not occur, it may be evidence that a reversal in the larger price trend may be in progress or provides evidence of a stronger bias at work in market. When price fails to make it to the medial line from either side, it is often an expression of the relative enthusiasm of buyers and sellers and may predict the next major direction of prices. If prices fail to reach the median line while above the median line, it is a bullish and failing to reach the median line from below is bearish.

Operating Andrews' Pitchfork tool is similar to drawing triangle. Left-click at the first point, hold down and drag to the second point, then release mouse button and drag to the third point and click once. The controlling triangle will become the pitchfork.

### Cycles tool

To use time cycles tool, click on the cycles drawing tool button in the toolbar then click at the starting point of the cycle and drag to the end of the cycle. These two control points control the interval between the cycle lines. When you release the mouse button you will get a series of parallel lines with equal interval in between them.

## Line study properties window

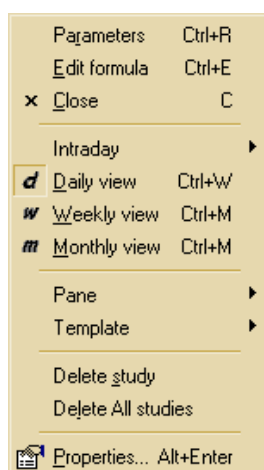


In the study properties window you can select start and end coordinates as well as line colours and styles. You can also enable automatic left- or right- line extension so that line will be extended when new quotes will be available.

There are following fields available:

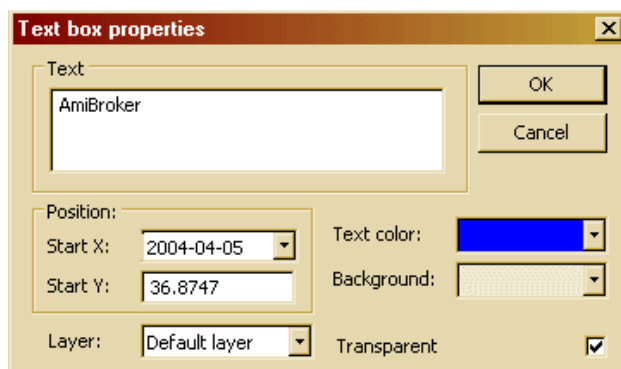
- **Start X, Start Y, End X, End Y** – study start and end coordinates
- **Third X, Third Y** – visible only for TRI-POINT studies like triangle, pitchfork – the coordinates of 3rd control point of the study
- **Lock position** – if this field is marked it's impossible to change the position of the study with use of mouse
- **Color** – allows you to change the study color
- **StudyID** – defines Study ID which allows you to refer to the study from AFL formula. The detailed information is available in [Using studies in your AFL formulas](#) chapter.
- **Layer** – indicates the layer that the study is placed on. To learn more about layers read [Working with layers](#).
- **Thick / Dotted line** – these options allow you to change the format of the study
- **Left / Right Extend** – you can choose whether line is extended

Line study properties window is accessible from chart window's right mouse button menu. When you click on a study line with a right mouse button the following menu appears:



Simply choose **Properties** to show the line study window.

## Text box properties window



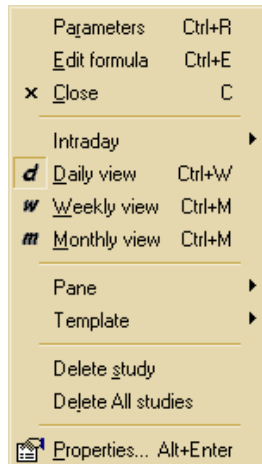
In the text box properties window you can change the text displayed in the box, select start co-ordinates as well as text and background colours and transparent style.

There are following fields available:



- **Start X, Start Y** – text coordinates
- **Color** – allows you to change the color of the text
- **Background Color** – allows you to change the color of the background
- **Layer** – indicates the layer that the text is placed on. To learn more about layers read [Working with layers](#).

Text box properties window is accessible from right mouse button menu. When you click on a text box with a right mouse button the following menu appears:



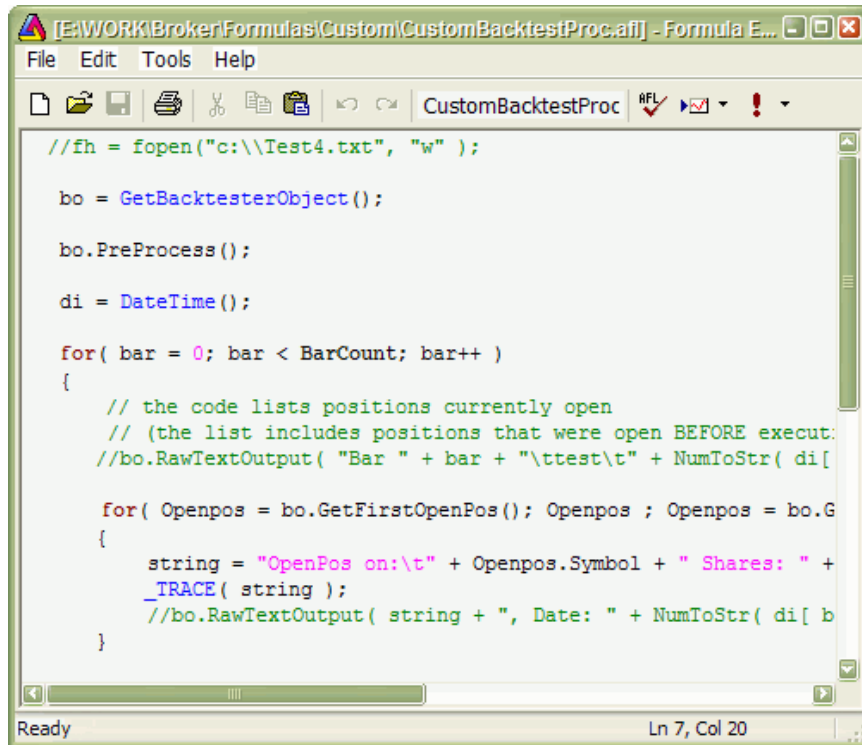
Simply choose **Properties** to show the text box properties window.

In the text box properties window you can change the text displayed in the box, select start co-ordinates as well as text and background colours and transparent style.

## Formula Editor

AFL Formula Editor features user-definable syntax highlighting, context-sensitive formula reference help, enhanced error reporting, automatic statement completion and parameter information technology (similar to Intellisense(tm) featured in Microsoft Developer studio), support for editing multiple files at once, and is multi-monitor friendly.

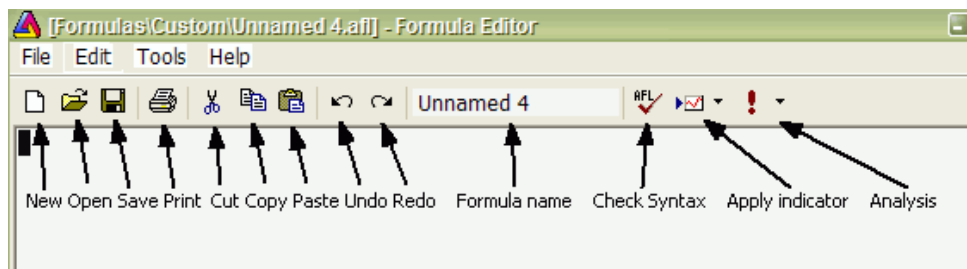
These features greatly simplify writing formula and provide instant help so time needed to write formula decreases significantly.



## Menu

Formula Editor menu options are described in detail in [Menus: Formula Editor](#) chapter of the guide.

## Toolbar



The Formula Editor toolbar provides the following buttons:

- **New** – clears the formula editor window
- **Open** – opens the formula file
- **Save** – saves the formula under current name
- **Print** – prints the formula
- **Cut** – cuts the selection and copies to the clipboard
- **Copy** – copies the selection to the clipboard
- **Paste** – pastes current clipboard content in the current cursor position
- **Undo** – un–does recent action (multiple–level)
- **Redo** – re–does recent action (multiple–level)
- **Formula Name** – an EDIT field that allows to modify the formula file name, once you change the name here and press **Save** button the formula will be saved under new name and the change will be reflected in editor CAPTION BAR.

- **Check syntax** – checks current formula for errors
- **Apply indicator** – saves the formula and applies current formula as a chart/indicator ONCE
- **Analysis** – saves the formula and selects it as current formula in Automatic Analysis window and repeat most recently used Analysis operation (i.e. Scan or Exploration or Backtest or Optimization)

## Usage

Typical use of Formula Editor is as follows:

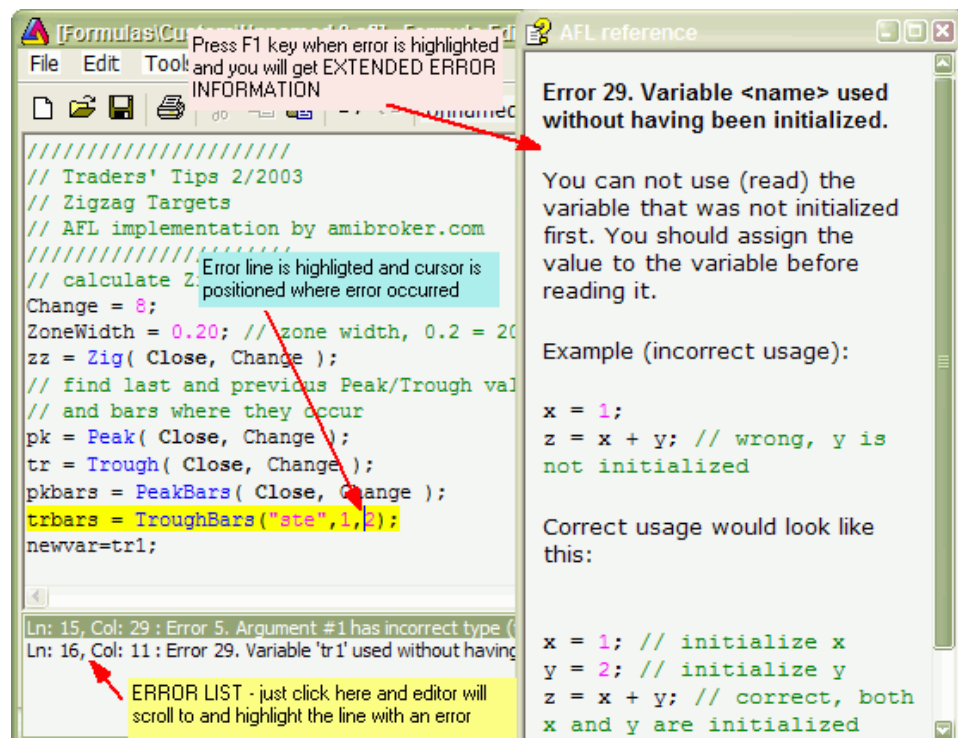
- open Formula Editor
- type the formula
- type meaningful name that describes the purpose of you code into **Formula Name** field
- click **Apply indicator** button (if you have written indicator code)  
.. or..  
click **Analysis** button to display Automatic Analysis window (when you have written exploration/scan or trading system)

## Syntax highlighting

AmiBroker's AFL editor features user-definable syntax highlighting that automatically applies user-defined colors and styles to different language elements like functions and reserved variable names, strings, numbers, comments, etc. This feature greatly simplifies code writing. You can modify coloring scheme in [Preferences window](#).

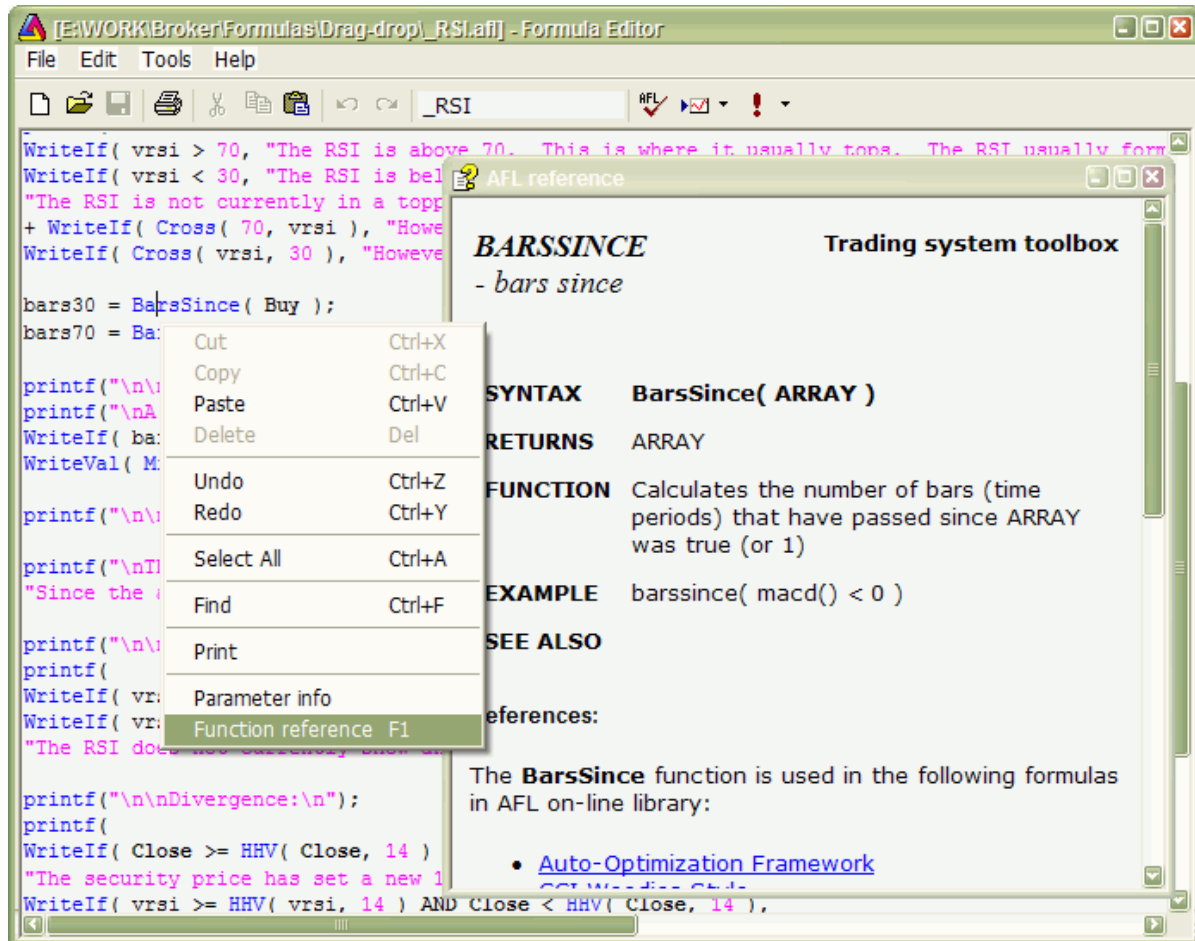
## Enhanced error reporting

When you make an error in your formula, AmiBroker's enhanced error reporting will help you to locate and fix an error by highlighting the place where error occurred and displaying extended error description with the examples of common mistakes and advice how to fix them.



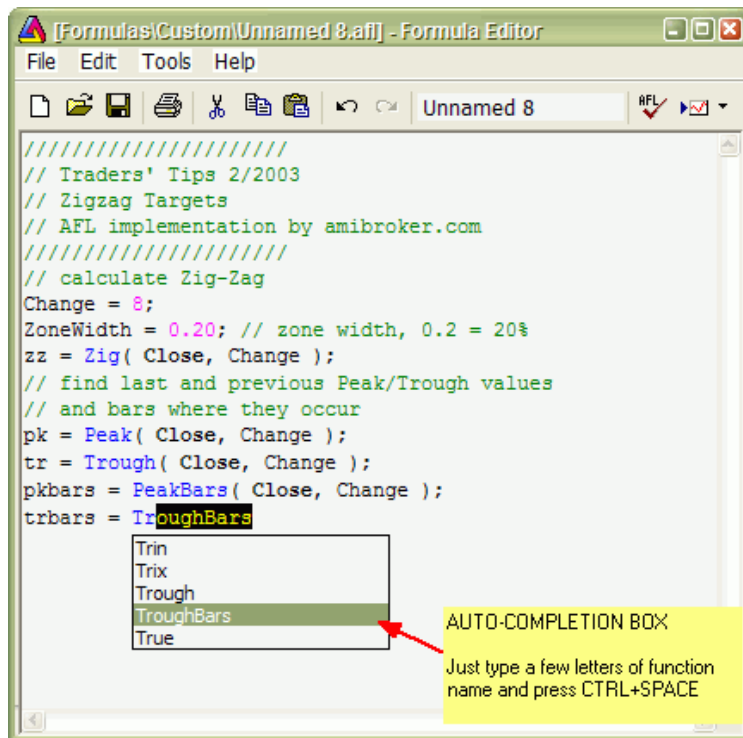
## Context help

You can quickly display relevant AFL function reference page if you press **F1** key or choose "Function reference" from the context menu while the caret is inside or right after function name as shown in the picture below:



#### Automatic statement completion

The automatic completion feature (available when you press **CTRL+SPACE** key combination) finishes typing your functions and reserved variables for you, or displays a list of candidates if what you've typed has more than one possible match. You can select the item from the list using up/down arrow keys or your mouse. To accept selection press RETURN (ENTER). You can also type immediately space (for variables) or opening brace (for function) and AmiBroker will auto-complete currently selected word and close the list. To dismiss the list press ESC key.



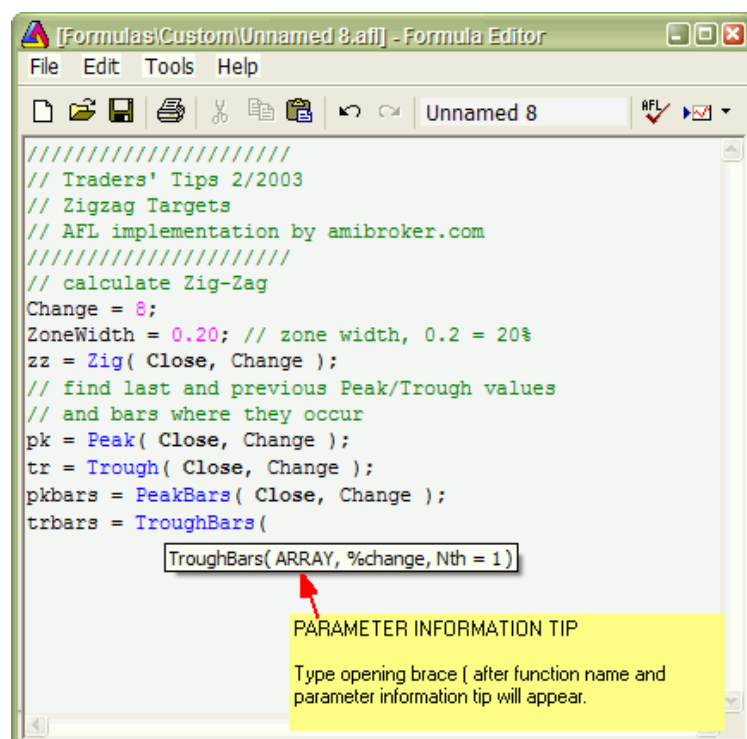
### Parameter Information

When you are typing a function, you can display a Tool Tip containing the complete function prototype, including parameters. The **Parameter Info** Tool Tip is also displayed for nested functions. With your insertion point next to a function, type an open parenthesis as you normally would to enclose the parameter list.

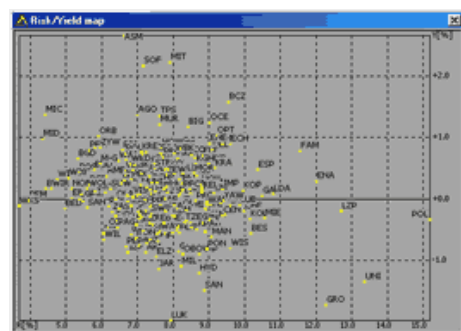
AmiBroker displays the complete declaration for the function in a pop-up window just under the insertion point.

Typing the closing parenthesis dismisses the parameter list.

You can also dismiss the list if you press arrow up/down key, click with the mouse or press RETURN.



### Risk–Yield Map window

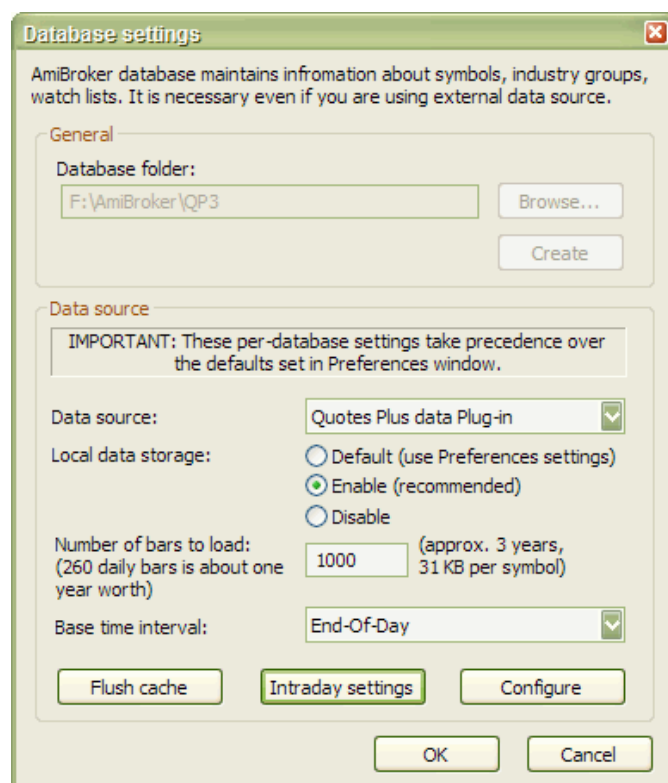


This map provides fast information about risk and possible yields. Yield is the average weekly percentage return while Risk is a standard deviation of percentage weekly returns. On the X axis risk is presented and on Y axis – yield. Thus in the upper part of the map we have got symbols with giving best yield, with risk increasing from left to right side of the map.

Selected symbol is marked with a different color, and you can zoom the part of the map by pressing left mouse button and marking rectangle to zoom in.

## Database Settings

This window allows you to define per-database settings. It is accessible via File->Database Settings menu.



**IMPORTANT:** These per–database settings in this window take precedence over default values definable in [Preferences](#) window. See explanation in [Tutorial: Understanding database concepts](#).

The database settings window is divided into two parts: *General* and *Data source*

*General* settings part are enabled **only at the database creation time (File→New database)**, once database is created these controls become disabled.

- **Browse...** – allows to browse for folder where new database should be created.
- **Create** – clicking on this button creates the database inside the folder specified in **Database folder** edit field.

For more details about creating new database working with particular data source please check [Tutorial section](#).

*Data source* part becomes enabled once database is created and it can be used to modify settings for already existing databases (via **File→Database Settings** menu). The following controls are available:

- **Data source:** defines data source, this can be either
  - ♦ (local) – it means that no external source is used and data are maintained by AmiBroker itself. Such database can be updated either using [AmiQuote \(Tools→Auto–update quotes\)](#) or using ASCII import – [Import Wizard](#), [Metastock importer](#), or [script](#).
  - ♦ external data source (one of: eSignal, myTrack, QuoteTracker, Quotes Plus, TC2000/TCNet, FastTrack, Metastock) – it means that data are retrieved directly from external database / data source. Such database is updated automatically via plugin and does not require any user action in AmiBroker. For example if you use TC2000 as a data source all data that are present in TC2000 system become automatically available in AmiBroker. For more details please read [Tutorial: Understanding database concepts](#).

- **Local data storage:** decides if data from **external data source** should be stored/cached also in AmiBroker's own files. If "Enabled" then external data are cached in local files. If "Disabled" then local files do not store external data. Switching this to "Enabled" is **required** for most **real-time** data sources as eSignal, myTrack, QuoteTracker. This setting has no effect if data source is set to (local).
- **Number of bars to load** – defines how many bars should be loaded from **external data source** and kept in AmiBroker. Examples: 10–years EOD: 2600, 60–days intraday 1–minute: 30000 (approx). This setting has no effect if data source is set to (local).
- **Base time interval** – defines what 'base' bar interval is used in this database. For real-time data sources this should be set once at the database creation time. This is so because real-time sources need to collect RT ticks and pack them (time-compress) into interval bars. This setting defines the minimum 'grain'. For EOD sources it is (End-of-day (daily)). For real-time sources this should be 1–minute or higher. For some real-time sources (like eSignal) this can be also set to tick, 5–sec or 15–sec.  
Please note also that you **won't be able to use intraday charting and/or analysis** until base time interval is set to something below end-of-day interval (it can be 1–minute for example). For more details please read [Tutorial: Basic charting guide](#).
- **Flush cache** – allows to force cache flushing and force retrieving fresh data from the plugin
- **Configure** – allows to display data source specific configuration dialog see [Tutorial section](#) for details on configuring various data sources.
- **Intraday settings** – allows to define per-database settings for intraday databases (see below)

## Intraday Settings window

- **Filtering** – this provides control over the display of intraday data. AmiBroker collects all the data but displays only those data which are inside selected trading hours start–end time. Please note that this affects all charts and windows except [Quote Editor](#) that always displays all available data.



**Show 24 hours trading (no filtering)** – all data are displayed (no filtering at all)

**Show day session only** – only the data between day session (RTH) start and end times are displayed

**Show night session only** – only the data between night session (ETH) start and times are displayed

**Show day and night session only** – only the data between either day session start/end time or night session start/end time are displayed

**Filter weekends** – when checked AmiBroker collects but does not display data from weekends. When unchecked those data are collected and displayed.

- **Trading hours Start / End** – defines trading hours start and end times for day (RTH) and night (ETH) sessions separately (see above). Please note that the times should be specified in your local time zone.
- **Daily time-compression uses** – this decides how AmiBroker performs intraday to daily time compression

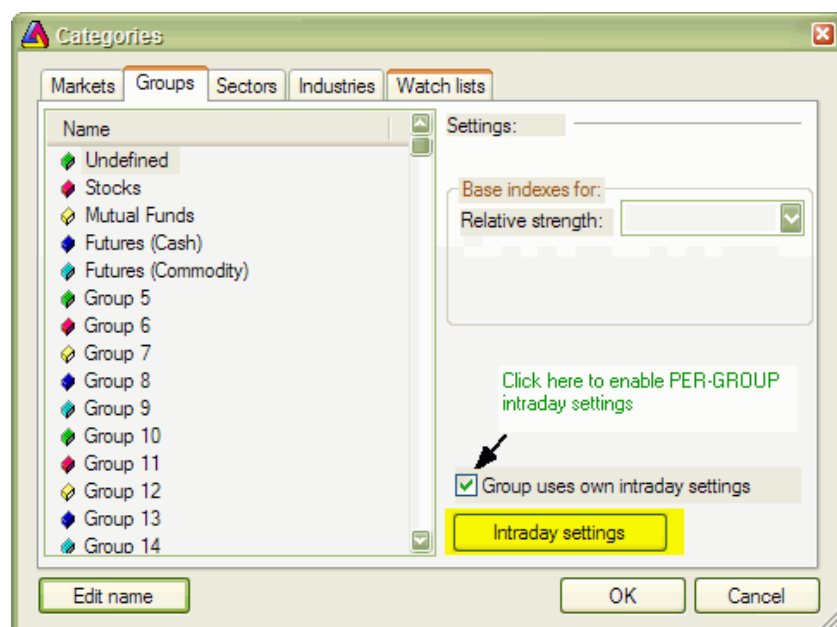
**Exchange time** – daily data are constructed from intraday bars starting from 00:00 and ending at 23:59 in the EXCHANGE (or data source) TIME ZONE

**Local time** – daily data are constructed from intraday bars starting from 00:00 and ending at 23:59 in the LOCAL (computer) TIME ZONE

**Day/Night session times as defined above** – daily data are constructed from the intraday bars that start at the start time of night session (previous day) and end at the end time of day session)

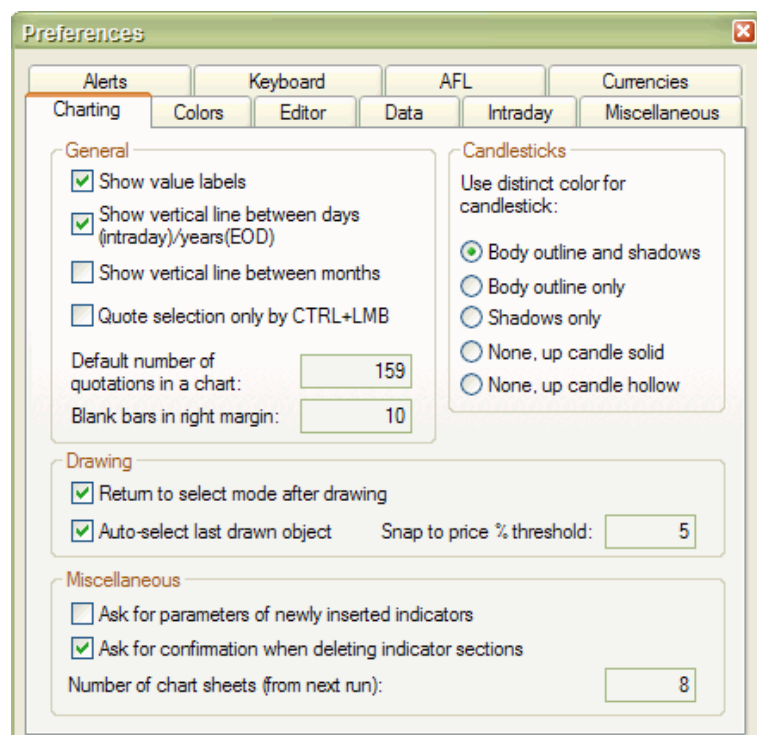
- **Time shift** – is the time difference (in hours) between your local time zone and the exchange time zone
- **Allow mixed EOD/Intraday data** – it allows to work with database that has a mixture of intraday and EOD data in one data file. If this is turned on then in intraday modes EOD bars are removed on-the-fly and in daily mode EOD bars are displayed instead of time compressed intraday or if there is no EOD bar for corresponding day then intraday bars are compressed as usual. This mode works in conjunction with new versions of plugins that allow mixed data. Mixed mode is now supported by MarketCast plugin (1.1 or higher)(Australia) and eSignal plugin (1.7.0 or higher) only. Mixed mode allows intraday plus very long daily histories in one database.

Note that **Intraday Settings** available from **Database Settings** dialog are **PER-DATABASE**. There is however also an option to define **PER-GROUP** intraday settings. To use PER-GROUP intraday settings you have to open [Symbol->Categories](#) window, switch to **Groups** tab and check "Group uses own intraday settings" box as shown in the picture below



Then you can click on **Intraday Settings** button to display per-group settings. Please note that each group in the category list can have its own individual settings so you can easily setup groups so they contain instruments traded in different hours. You can move symbols between groups using [Symbol->Organize assignments](#) dialog.

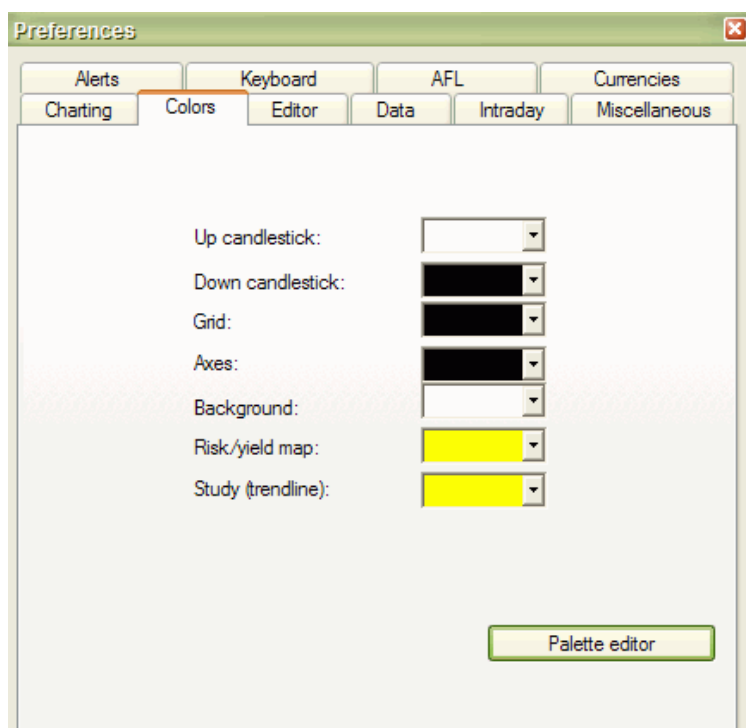
## Preferences window



**Charting tab** – allows you to modify charting options

- **Default number of quotations in a chart** – this sets the amount of bars initially displayed in the chart. (in other words it defines "normal" zoom range)
- **Blank bars in right margin** – defines how many blank bars are added in the right margin (past the last available quote). This blank margin allows you to project studies (trend lines for example) into the future
- **Quote selection only by CTRL+LMB** – this decides how vertical selection line is invoked. When this box is unchecked – single click on the chart causes display of the selection line, when this box is checked you have to hold down CTRL key while clicking to get the selection line
- **Show vertical line between days (intraday)/years(EOD)** – this decides if dotted vertical line is displayed on the chart to mark day (in intraday mode) or year (in EOD mode) boundaries
- **Show value labels** – this decides if value labels for indicator / price chart lines should be displayed. See [basic charting guide](#) for explanation what value label is.
- **Candlesticks** – this setting provides detailed control over the appearance of candlesticks. The distinct color may be used to draw part of the candlestick or entire candle may be drawn in the same color as its interior.
- **Drawing**
  - ◆ **Return to select mode after drawing** – when checked current tool is deactivated after drawing and select mode is entered, when unchecked currently selected drawing tool remains active after drawing (allows to plot one study after another, note that the same effect can be achieved even if this box is checked – it is enough to hold down SHIFT key while drawing and the tool will remain active)
  - ◆ **Auto-select last drawn object** – this useful feature automatically selects recently drawn object. This allows to hit ALT+ENTER to display properties box immediately without need to click on the study, and allows to Copy the study via CTRL+C also without additional click
  - ◆ **Snap to price % threshold** – defines how far price 'magnet' works, it will snap to price when the mouse is nearer than % threshold from H/L/C price
- **Miscellaneous**
  - ◆ **Ask for parameters of newly inserted indicators** – when checked AmiBroker will automatically display [Parameters](#) window each time you insert new indicator or overlay one indicator over another.
  - ◆ **Ask for confirmation when deleting indicator sections** – when checked AmiBroker will ask you to confirm deletion of any overlaid indicator section (applies to indicators created via drag-and-drop). Please note that deletion of indicator section modifies the underlying formula. More on this in [Tutorial: Drag-and-drop](#)
  - ◆ **Max number of chart sheets** – defines how many chart sheets (tabs) should be available. More information on chart sheets is in the [Tutorial section here](#). Note that this setting will take effect after restart.

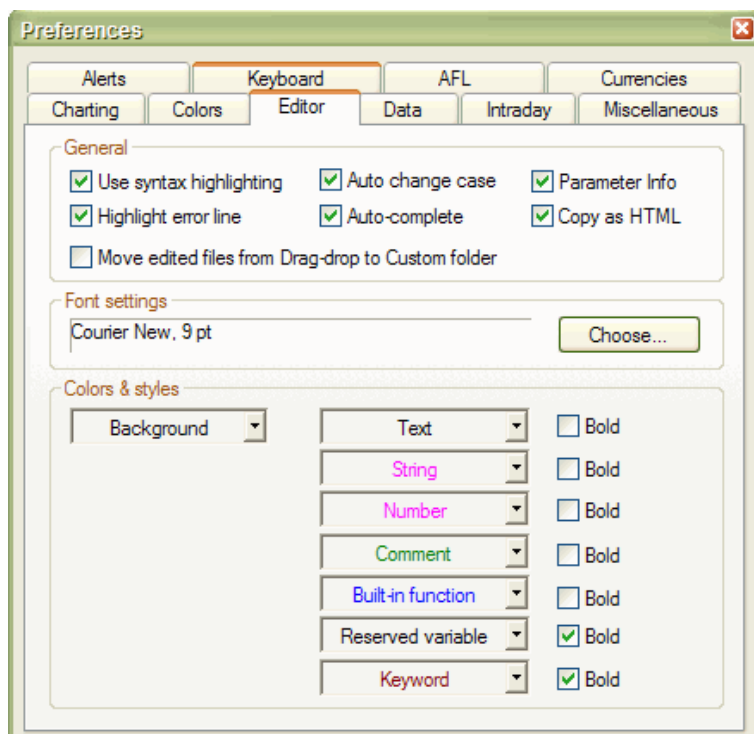
•



**Color tab** – allows to define colors for particular chart element.

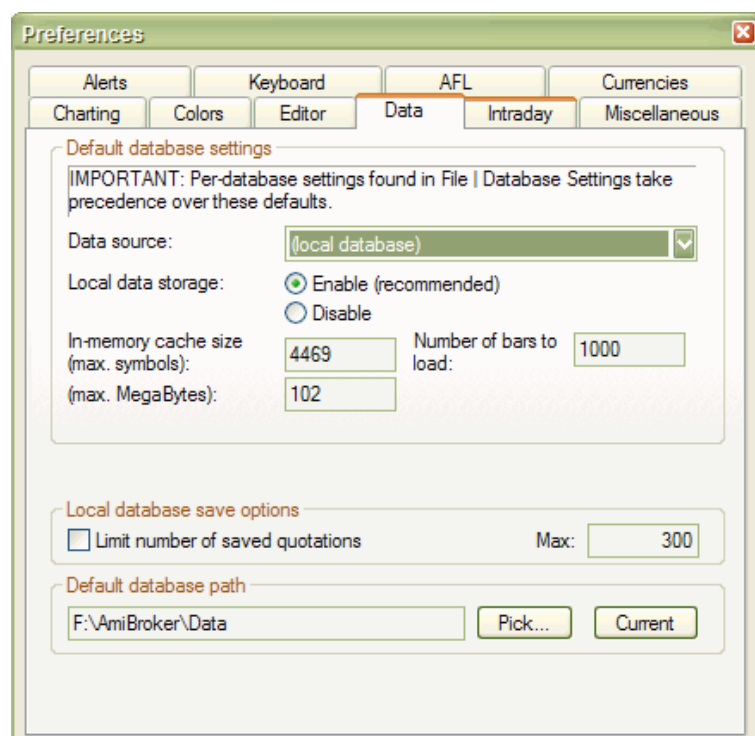
The controls provide user definable color selection for charts, grid & background.

**Palette editor** – allows to modify custom colors that can be referenced later via colorCustom0..colorCustom15 constants



**Editor tab** – controls the appearance and features of [AFL editor](#).

- **Use syntax highlighting** – when checked editor automatically colorizes your code (different colors/styles for functions, constants, numbers, etc)
- **Auto-change case** – when checked the function and reserved variable names are automatically capitalised so if you type bARSSince, editor will change it to BarsSince
- **Auto-complete** – when checked you will be able to use [auto-completion feature](#) (CTRL+SPACE will auto-complete the word)
- **Parameter info** – when checked the editor will display [parameter information tooltip](#) when you type a function name and opening brace
- **Highlight error line** – when checked the formula editor marks the line of code that contains an error with a yellow background (Windows 2000 and XP only)
- **Copy As HTML** – when checked the AFL editor on Edit->Copy / Cut command puts not only plain text and RTF formats to the clipboard but also HTML and DwHTML (Dreamweaver HTML) formats allowing pasting syntax-colored code to Macromedia Dreamweaver and other HTML-aware applications. Note: rarely (on very few machines) turning this on may cause problems with pasting to Outlook.
- **Move edited files from Drag-drop to Custom folder** – when checked the editor will automatically move manually edited formulas created by drag-and-drop mechanism inside hidden 'Drag-drop' subfolder to 'Custom' subfolder.
- **Font settings** – allows you to define AFL editor font face and size
- **Colors and styles** – allows you to define what colors and styles will be used to mark certain language elements when syntax highlighting is ON.

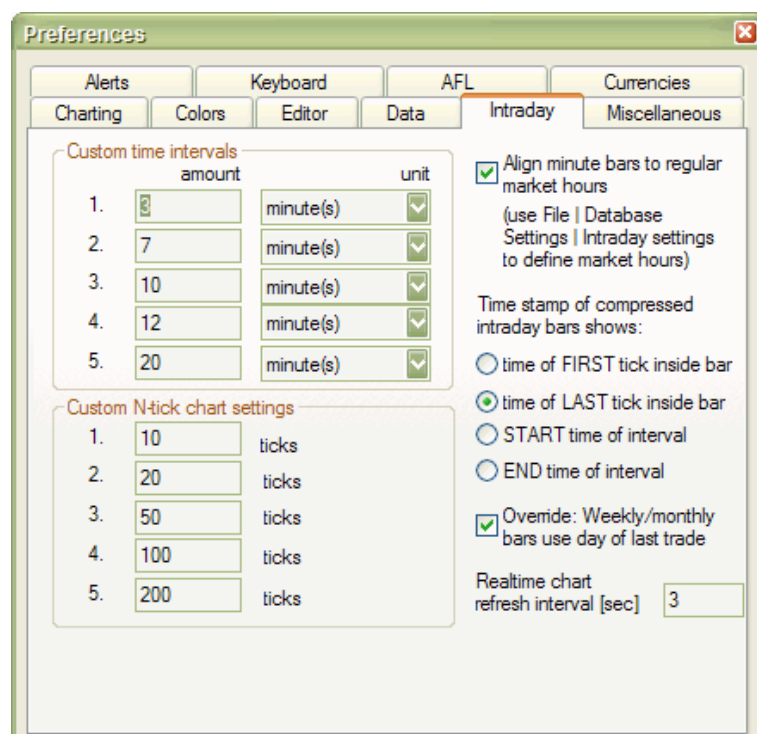


**Data tab** – allows you to define default, global values for all databases.

**IMPORTANT:** some of these settings may **get overwritten by PER-DATABASE settings in File->Database Settings window**. See explanation in [Tutorial: Understanding database concepts](#).

- **Data source:** defines default data source (for databases that do not specify other source in File->Database Settings)

- **Local data storage:** default setting for **external** databases (this setting gets overwritten by File->Database Settings). If "Enabled" then external data are cached in local files. If "Disabled" then local files do not store external data.
- **In-memory cache (max. symbols)** – defines how many symbols data should be kept in RAM (for very fast access) – this works together with the next setting
- **In-memory cache (max. MegaBytes)** – defines how many MB of RAM should be used for temporary data cache (for very fast access)
- **Number of bars to load** – default setting for **external** databases (this setting gets overwritten by File->Database Settings). Defines how many bars should be loaded from external data source and kept in AmiBroker. Examples: 10-years EOD: 2600, 60-days intraday 1-minute: 30000 (approx)
- **Limit number of saved quotations** – if this option is ON AmiBroker will save database with limited number of quotations. This prevents the database from growing too much
- **Max. number of saved quotations** – this is the limit itself. Preferable 300 or higher for EOD databases, 3000 or higher for intraday
- **Default database path** – this defines the path to the **database that is loaded on startup. If such database does not exist it will be re-created at startup time.**



**Intraday tab** – provides settings for intraday charting

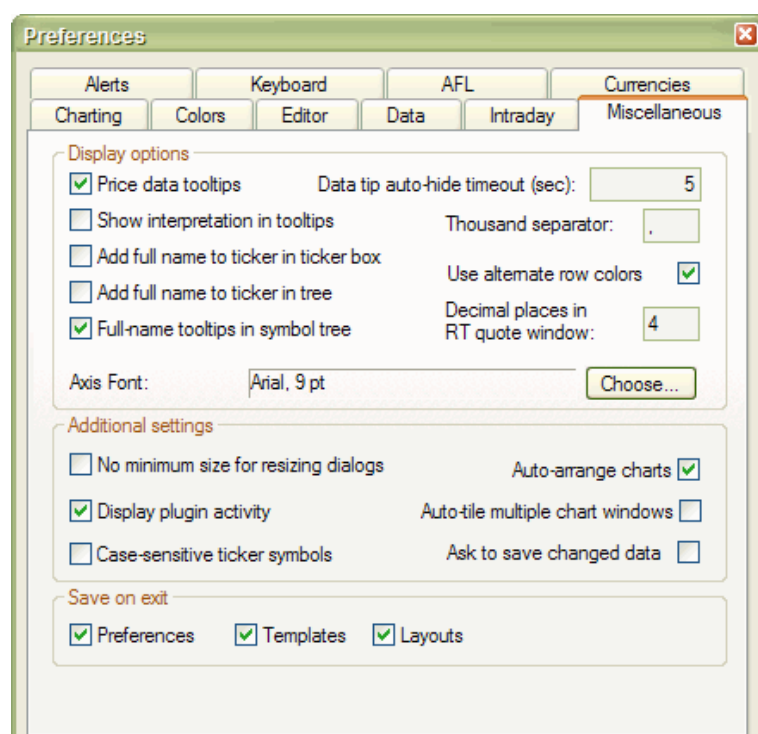
- **Custom time intervals** – allow to define your own N-minute or N-hours intervals (available later from View->Intraday menu)
- **Custom N-tick chart settings** – allow to define your own N-tick charts (available later from View->Intraday menu)
- **Align custom minute bars to regular market hours** – when checked AmiBroker will trim pre-market custom interval bar so new bar will begin exactly when trading hours start. Trading hours can be set per-database in File->Database Settings->Intraday settings. Let's say that we have 45-minute bars. Without this setting we would get bars starting at 9:00, 9:45, 10:30, 11:15 etc. When this is turned on and trading starts at 9:30 we have guarantee that bars will be aligned to 9:30: 8:45, 9:30, 10:15, 11:00

- **Time compressed bars shows:**
  - ◆ **time of FIRST tick inside bar** – when selected the bar gets the time stamp of the very first trade inside given time slot (bar)
  - ◆ **time of the LAST tick inside bar** – when selected the bar gets the time stamp of the very last trade inside given time slot (bar)
  - ◆ **START time of the interval** – when selected the bar is time-stamped with start time of the time slot (bar). Lets say that 30 minute bar covers 9:00:00..9:29:59. When this is selected AmiBroker will display time of this bar as 9:00
  - ◆ **END time of the interval** – when selected the bar is time-stamped with start time of the time slot (bar). Lets say that 30 minute bar covers 9:00:00..9:29:59. When this is selected AmiBroker will display time of this bar as 9:29:59
- **Realtime chart refresh interval** – defines interval between automatic chart refreshes in real-time mode. By default charts are refreshed every 3 seconds but in very volatile market you may prefer to set it to 1, so charts are refreshed every second in real-time mode.

**New in 4.90:** To enable 'every tick' chart refresh in **Professional** Edition, go to Tools->Preferences, Intraday tab and enter ZERO (0) into "Intraday chart refresh interval" field. (note Standard Edition won't allow to do that).

Once you enter zero, AmiBroker will refresh all charts with every new trade arriving provided that the formulas you use execute fast enough. If not, it will dynamically adjust refresh rate to maintain maximum possible refresh rate without consuming more than 50% of CPU (on average). So for example if your charts take 0.2 sec to execute AmiBroker will refresh them on average 2.5 times per second.

Note: built-in Windows Performance chart shows cumulated CPU consumption for all processes, to display PER-PROCESS CPU load use SysInternals free software  
<http://www.sysinternals.com/Utilities/ProcessExplorer.html>



- **Price data tooltips**  
if checked small tooltips will appear when you hover over the chart displaying selected bar date, prices / indicator values
- **Show interpretation in tooltip**  
if checked data tooltips will include also interpretation text that is normally displayed in the [Interpretation window](#).
- **Data tip auto-hide timeout**  
defines how many seconds data tooltip should remain on the screen if you don't move your mouse.
- **Add full name to ticker in the ticker box**  
when checked the ticker box displays not only symbol but also full name of the issue
- **Add full name to ticker in the tree**  
when checked the workspace tree displays not only symbol but also full name of the issue
- **Full-name tooltips in symbol tree**  
when checked then full name of symbol is displayed in the tooltip that appears when you move your mouse over symbol in the symbol tree.
- **Data-tip auto-hide timeout**  
defines time in seconds how long data tooltip (that shows values of indicators) will be displayed when mouse cursor does not move
- **Thousand separator**  
defines thousand separator for number displayed on charts and all list-views.
- **Decimal places in RT quote window**  
defines how many decimal places should be displayed in Real Time quote window.
- **Axis font**  
defines font face and size to be used for chart axis and text tool
- **No minimum size for resizing dialogs**  
when checked it allows to size dialogs below the minimum size (so some controls become invisible)
- **Display plugin activity**  
when checked AmiBroker displays information about accessing data plugin in the status bar
- **Case sensitive ticker symbols**



when checked ticker symbols are case sensitive. In other words INTC and Intc and iNTc are considered DIFFERENT. This is required for some Canadian symbols for example. Please use with caution. If your exchange do not use case-sensitive tickers please make sure it is UNCHECKED.

- **Auto-arrange charts**

if this option is on chart windows are scaled and arranged to fit the screen after every opening/closing chart window.

- **Auto-tile multiple chart windows**

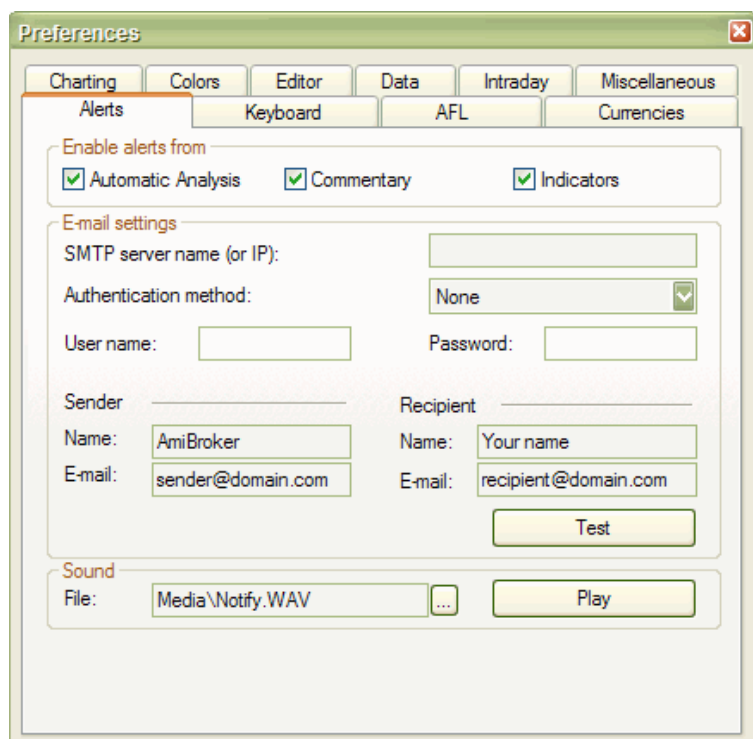
when checked multiple chart windows are always tiled **vertically** on every resize of the main application window.

- **Ask to save changed data**

when checked AmiBroker asks if you want to save modified data on exit. When unchecked AmiBroker saves modified data without asking.

- **Save on exit: Preferences, Templates, Layouts**

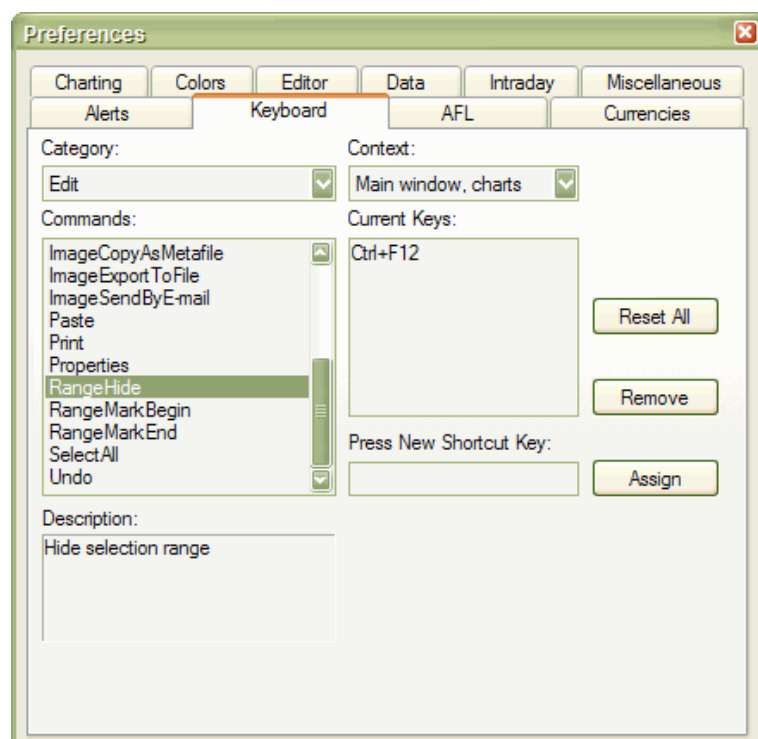
controls which settings should be saved automatically on exit



**Alerts tab** – It allows to define e-mail account settings, test sound output and define which parts of AmiBroker can generate alerts via AlertIF function.

**E-mail settings** page now allows to choose among most popular authorization schemes like: AUTH LOGIN (most popular), POP3-before-SMPT (popular), CRAM-MD5, LOGIN PLAIN.

**Enable alerts from** checkboxes allow you to selectively enable/disable alerts generated by Automatic analysis, Commentary/Interpretation and custom indicators.



### Keyboard tab

Allows you to define and/or modify keyboard shortcuts for menu items / commands.

#### **To assign a shortcut key**

On the Tools menu, click Preferences, and then click the Keyboard tab.

In the Categories list, select the menu that contains the command to which you want to assign the shortcut key.

In the Commands list, select the command to which you want to assign the shortcut key.

Put the cursor in the Press New Shortcut Key box, press the shortcut key or key combination that you want, and click Assign.

If you press a key or key combination that is invalid, no key is displayed. You cannot assign key combinations with ESC, F1, or combinations such as CTRL+ALT+DEL that are already being used by your operating system.

If you press a key or key combination that is currently assigned to another command, that command appears under Currently Assigned To.

#### **To delete a shortcut key**

On the Tools menu, click Preferences, and then click the Keyboard tab.

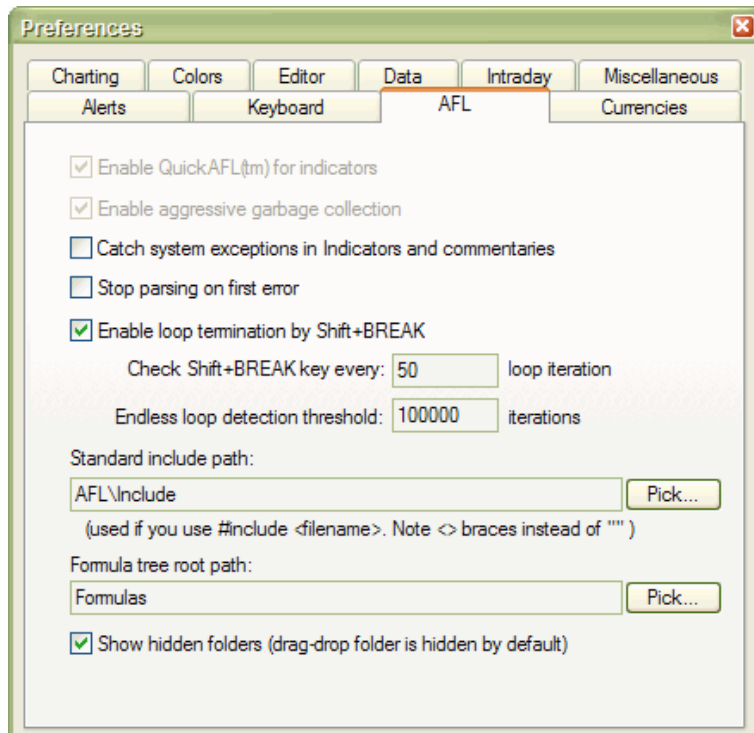
On the Categories, and Commands lists, select the location for the shortcut key you want to delete.

In the Current Keys list, select the shortcut key you want to delete and click Remove.

**To reset all shortcut keys to their default values**

On the Tools menu, click Preferences, and then click the Keyboard tab.

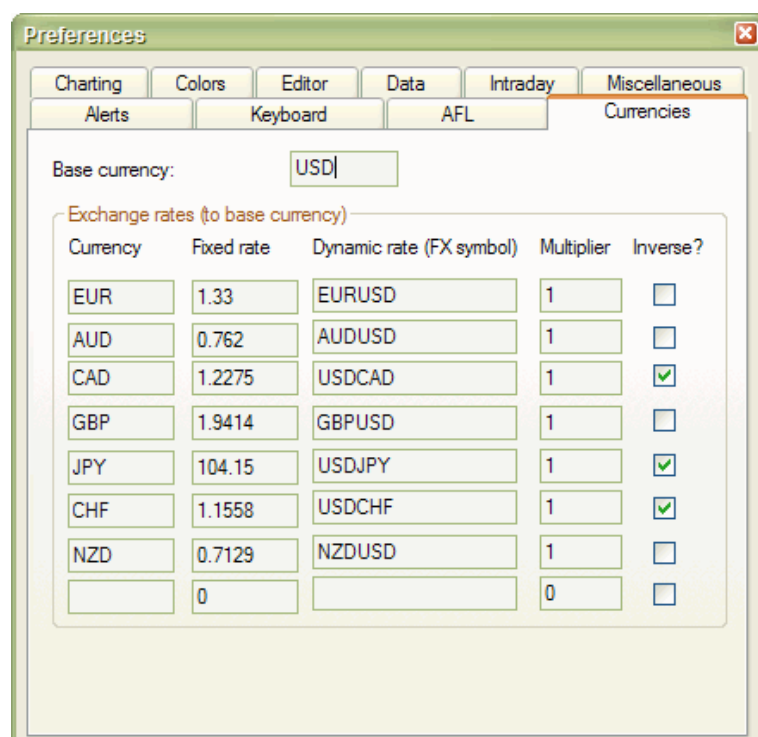
Click Reset All.

**AFL tab**

- **Catch system exceptions in Indicators and commentaries** – when checked all exceptions (run-time errors) are caught by the indicator drawing code, so no [Bug Recovery](#) window appears. Instead exception information is displayed inside chart pane. It is recommended to have this turned ON especially when you use real-time data
- **Stop parsing on first error** – when checked parser stops further code analysis on first encountered error so only one (first) error is displayed in the [formula editor](#) error list. If it is unchecked then parser will list all errors found. It is recommended to turn it off.
- **Enable loop termination by Shift-BREAK** – when checked AmiBroker will allow to break any for(), while() and do-while() loop by pressing and holding down SHIFT and BREAK(PAUSE) keys on your keyboard.
- **Check Shift+BREAK key every** – defines how often keyboard state should be checked when loop is executed. Note that specifying small values will make loop execution slower.
- **Endless loop detection threshold** – defines the number of loop iterations after which AmiBroker will terminate the loop with "Possible Endless loop detected" error message. This is useful in situations when the code has infinite loop (due to mistake of the formula author) because it won't allow AmiBroker to hang due to infinite looping
- **Standard include path** – the default path to use when #include statement uses < > braces instead of ""
- **Formula tree root path** – the root path of Formula file/directory tree displayed in the Charts tab of

Workspace window

- **Show hidden folders** – determines if formula tree should show subfolders with "hidden" attribute (drag-drop folder is created as "hidden" by the setup program)



### Currencies tab

This page allows to define base currency and exchange rates (fixed or dynamic) for different currencies. This allows to get correct backtest results when testing securities denominated in different currency than your base portfolio currency. For more details please check [Tutorial: Pyramiding and multiple-currency support in the backtester](#).

How does AB know whether I want the fixed or dynamic quote?

There are following requirements to use currency adjustments:

- Symbol→Information, "Currency" field shows currency different than BASE currency
- Appropriate currency (defined in Symbol) has matching entry in Preferences→Currencies page
- the dynamic rate "FX SYMBOL" defined in the preferences EXISTS in your database and HAS QUOTES for each day under analysis range.

What is "INVERSE" check box for in the preferences?

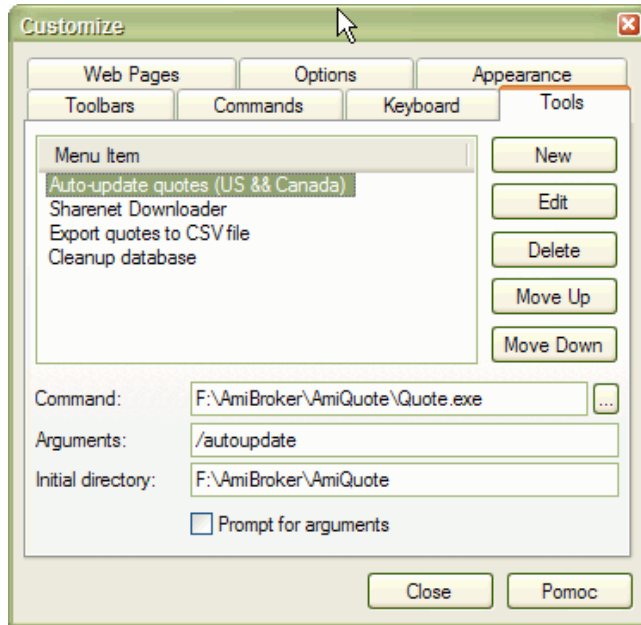
Let's for example take EURUSD.

When "USD" is your BASE currency then EUR exchange rate would be "straight" EURUSD fx (i.e. 1.3). But when "EUR" is your BASE currency then USD exchange rate would be INVERSE of EURUSD (i.e. 1/1.3). Opposite would be true with FX rates like USDJPY (which are already "inverse").

## Customize tools window

This dialog allows you to customize the User Interface. It can be invoked from *Tools*→*Customize* menu.

In "Tools" tab you define custom tool menu items:



You can launch executable files (.exe), script files (.js, .vbs), web pages (.html) and any other registered file types from the tools menu. In order to add a new tool you should open this dialog and click "New" button. Then enter the tool name, command (by hand or using file dialog) optional arguments and initial directory. If you check "Prompt for arguments" checkbox AmiBroker will ask for program's arguments each time

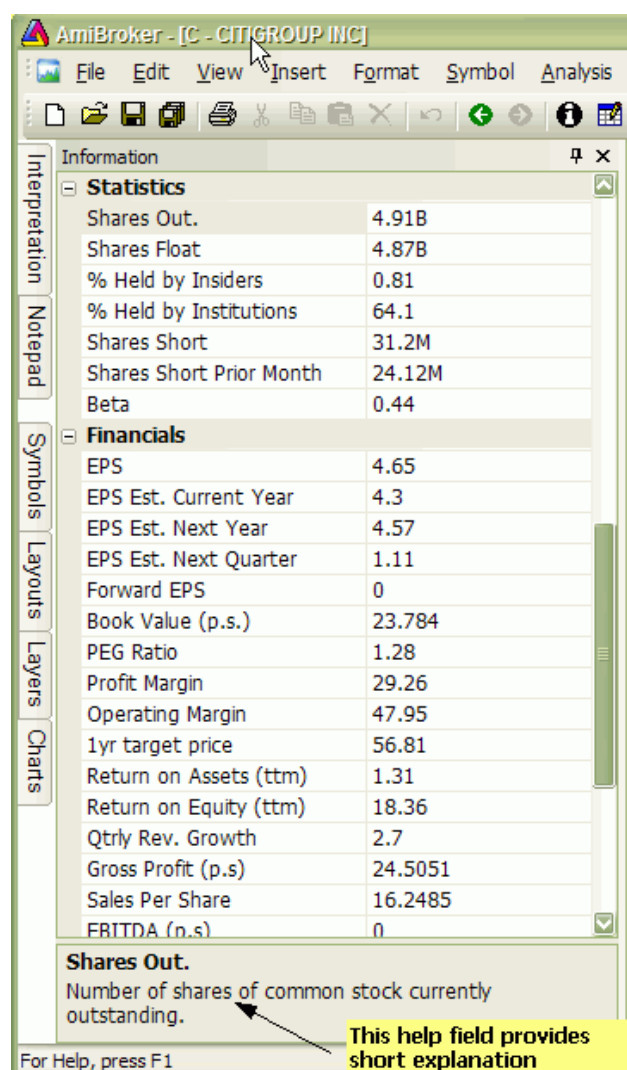
Other tabs provide UI customization features described in [Customize UI tutorial](#) section.

## Symbol tree window

In these windows we have got list of available symbol. Selecting one of them will refresh all opened charts and update information windows. Tree browser pane shows tickers belonging to different categories in separate leaves. This makes it easy to browse symbols by category.

This selection is global for the program i.e. all symbol functions will reference symbol selected in this window.

## Information window



This window allows you to display and edit preferences of the symbol.

- **Symbol**

The short name, used in 'Select' window and with quotation import functions. If you use them, please check if ticker given in this field is the same as used in your quotation datasource

- **Alias**

The alternative ticker name. It will be useful if you e.g. get the realtime quotes and backfill from two separate datasources, that use different ticker names.

- **Full name**

Official version of firm name

- **Code**

Symbol code number

- **Web ID**

Symbol Web ID – can be used when you define [Profile view](#)

- **Address**

Corporation address

- **Issue**

Total number of shares

- **Nominal value**

- **Book value**

- **Currency**

- **Market**

Indicates which market the symbol belongs to.

- **Industry**

Indicates which industry the symbol belongs to.

- **Group**

Indicates which Group the symbol belongs to.

- **Round lot size**

Various instruments are traded with various "trading units" or "blocks". For example you can purchase fractional number of units of mutual fund, but you can not purchase fractional number of shares. Sometimes you have to buy in 10s or 100s lots. AmiBroker now allows you to specify the block size on global and per-symbol level.

You can define per-symbol round lot size in the Symbol->Information page. The value of zero means that the symbol has no special round lot size and will use "Default round lot size" (global setting) from the Automatic Analysis settings page. If default size is set also to zero it means that fractional number of shares/contracts are allowed.

- **Tick size**

This setting controls the minimum price move of given symbol. You can define it on global and per-symbol level. As with round lot size, you can define per-symbol tick size in the Symbol->Information page (pic. 3). The value of zero instructs AmiBroker to use "default tick size" defined in the Settings page (pic. 1) of Automatic Analysis window. If default tick size is also set to zero it means that there is no minimum price move.

Note that the tick size setting affects ONLY trades exited by built-in stops and/or ApplyStop(). The backtester assumes that price data follow tick size requirements and it does not change price arrays supplied by the user.

So specifying tick size makes sense only if you are using built-in stops so exit points are generated at "allowed" price levels instead of calculated ones. For example in Japan – you can not have fractional parts of yen so you should define global ticksize to 1, so built-in stops exit trades at integer levels.

- **Margin deposit** – explained in [Backtesting systems for futures contracts](#)

- **Point value** – explained in [Backtesting systems for futures contracts](#)

- **Continuous quotations**

Enables continuous trading for this symbol (this enables candlestick charts and manual entry open/high/low/volume controls and candlestick charts), otherwise symbol is traded with price fixing

- **Index**

Specifies if symbol belongs to *Indexes* category.

- **Favourites**

Specifies if symbol belongs to *Favourites* category.

- **Use only local database for this symbol**

Indicates that symbol is not updated via the plugin in real-time database. This field is checked by default if the symbol is added into realtime database as a result of import from ASCII file (also AmiQuote download). This setting allows you to keep additional symbols in the database and prevent plugin from overwriting the imported data.

For explanation of Fundamental data fields please read "[Tutorial: Using fundamental data](#)" chapter of this guide.

## Notepad window

Notepad window (that you can show/hide using **View->Notepad** menu) that allows to store free-text notes about particular security. Just type any text and it will be automatically saved / read back as you browse through symbols. Notes are global and are saved in "Notes" subfolder as ordinary text files.

Notes can be also read and written to using AFL language NoteGet and NoteSet functions.

```
NoteGet( "Symbol" );
```

– retrieves note linked to "symbol". If symbol is "" (empty string) then current symbol is used

```
NoteSet( "Symbol", "Text..." );
```

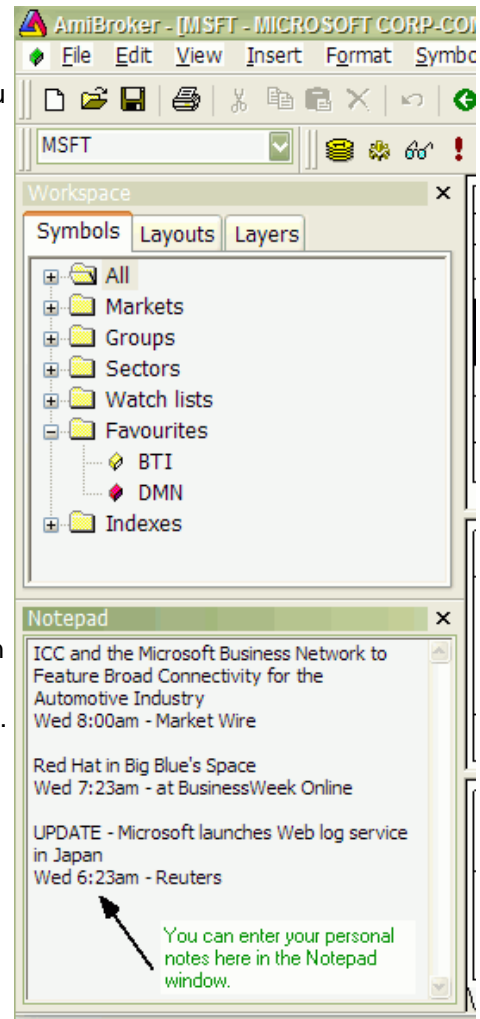
– sets text of the note linked to "symbol".

If symbol is "" (empty string) then current symbol is used.

If you overwrite note from AFL level that is opened at the same time in Notepad editor the editor will ask you (when you switch the focus to it) if it should reload new text or allow to save your manually entered text.

Example:

```
NoteSet("AMD", "Jun 15, 2004: AMD will deliver its  
first multi-core processors next year" );
```



## Quote Editor window



Quote Editor allows editing, deleting and adding quotes.



**To add new quote:**

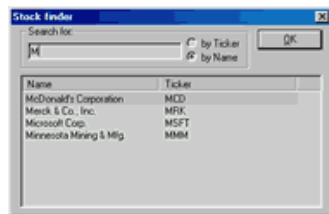
- select (new) entry
- enter date/time
- enter price data
- click on the list on the entry other than (new)

**To edit existing quote:**

- select quote from the list
- edit price data
- click on the list on the entry other than current

**To delete existing quote(s):**

- mark one or more quotes (multiple selection possible by holding down SHIFT or CTRL key)
- click "Delete" button

**Symbol Finder window (F3)**

Stock finder window allows you to quickly search the database for a symbol by typing the first letters of its full name or ticker. This feature is very useful when you don't know the ticker symbol. The symbol finder is accessible via *Edit*→*Find symbol*, *Symbol*→*Find* menus or by pressing F3 key.

To find a symbol just type one or more letters in the **Search for** box. Choose **by Name** if you want to perform full name search or choose **by Ticker** if you want to look up for the ticker. When you type the letters in the edit box appropriate symbols will appear in the list. You can click on the item to choose one or you can just press ENTER key to select the first one. Note that searching starts when the edit box contains at least 1 character – if it is empty no symbol is shown in the list.

**Financial data window**

	1st Quarter	2nd Quarter	3rd Quarter	4th Quarter
Year:	1996	1996	1995	1995
Sales income:	34076	20326	34522	36277
Earnings before tax (EBT):	6834	14583	4627	21520
Earnings after tax (EAT):	5312	9929	3198	13390
Price-to-Earnings ratio:	8.709	Price-to-Book Value ratio:	1.909	

This window provides fundamental financial data of corporation as:

- sales income
- earnings before taxes (EBT)
- earnings after taxes (EAT)

These are given in a quarter base.

Program will compute P/E (Price to Earnings ratio) and EPS (Earnings Per Share) indicators from given data.

## Profile view pane

Profile view pane is an embedded web-browser that allows you to view both on-line company's profiles as well as pages stored locally on your hard disk.



You can define URL-template for viewing on-line (or off-line) company's profiles. These URL-templates are market-based, which means you can have different templates for each market. The template is then parsed to make actual URL to the web page. For example to see Yahoo's profiles page you can use following URL template: `http://biz.yahoo.com/p/{t0}/{t}.html`. Symbols enclosed in brackets {} define fields which are evaluated in execution time. {t0} symbol is evaluated to the first character of the ticker name and {t} is evaluated to the whole ticker name. So if AAPL is selected AmiBroker will generate following URL from above template: `http://biz.yahoo.com/p/a/aapl.html`

Then AmiBroker uses built-in web browser to display the contents of the page.

Note that it is also possible to browse local files using this mechanism. Simply use following template URL (example, C: denotes drive): `file://C:\the_folder_with_profile_files\{t}.html`. You are not limited to HTML files, you can use simple TXT files instead.

## Special fields encoding scheme

As shown in above example template URL can contain special fields which are substituted at run time by values corresponding to the currently selected symbol. The format of the special field is {x} where x is describes field type. Currently there are three allowable field types: ticker symbol in original case {t}, ticker symbol in lowercase {s}, ticker symbol in UPPERCASE {S}, alias {a}, web id {i}. You can specify those fields anywhere within the URL and AmiBroker will replace them with appropriate values entered in the [Information window](#).

You can also reference to single characters of ticker, alias or web id. This is useful when given web site uses first characters of, for example, ticker to group the html files (Yahoo Finance site does that), so you have files

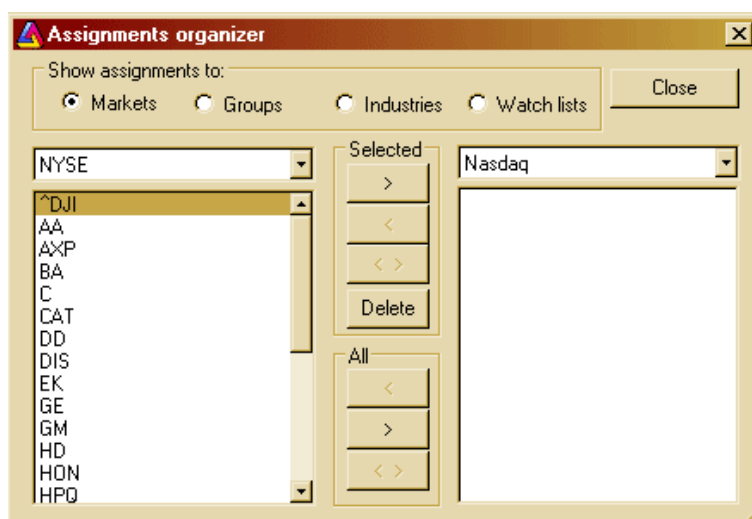
for tickers beginning with 'a' stored in subdirectory 'a'. To reference to single character of the field use second format style {*xn*} where *x* is field type described above and *n* is zero-based index of the character. So {a0} will evaluate to the first character of the alias string. To get first two characters of a ticker write simply {t0}{t1}.

Note about web id field: this new field in [Information window](#) was added to handle situations when web sites do not use ticker names for storing profile files. I found some sites that use their own numbering system so they assign unique number to each symbol. AmiBroker allows you to use this nonstandard coding for viewing profiles. All you have to do is to enter correct IDs in Web ID field and use appropriate template URL with {*i*} keyword.

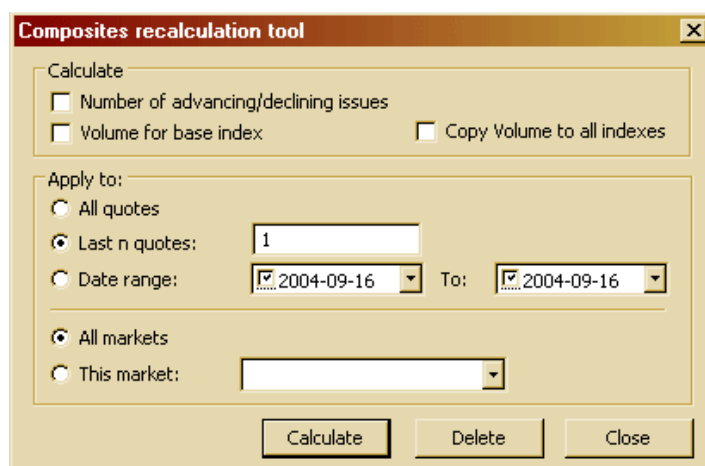
## Assignment organizer window

In order to make assigning the symbols to categories simpler and faster a new assignment organizer was developed. Now you can simply mark a group of symbols and quickly move them from one category to another.

**You can also delete multiple symbols using this window.** To do so, select one or more symbols from the left pane and click on "Delete" button.



## Composite recalculation window



This dialog allows automatic calculation of number and volume of advancing/declining/unchanged issues. Also possible in this dialog is calculation of volume numbers for indexes if imported incorrectly. Note well, that automatic recalculation of composite data makes only sense when you follow whole exchange (all symbols are included in your database) otherwise this calculation will give wrong results.

In order to calculate the composites in the database it's necessary to set the base index for the market, as it may happen that not all stocks are quoted every business day. AmiBroker checks the 'base index' quotations dates and tries to find the corresponding quotes of all the stocks belonging to that market, to find out how many issues advanced, declined and not changed at all.

To calculate composites you need to:

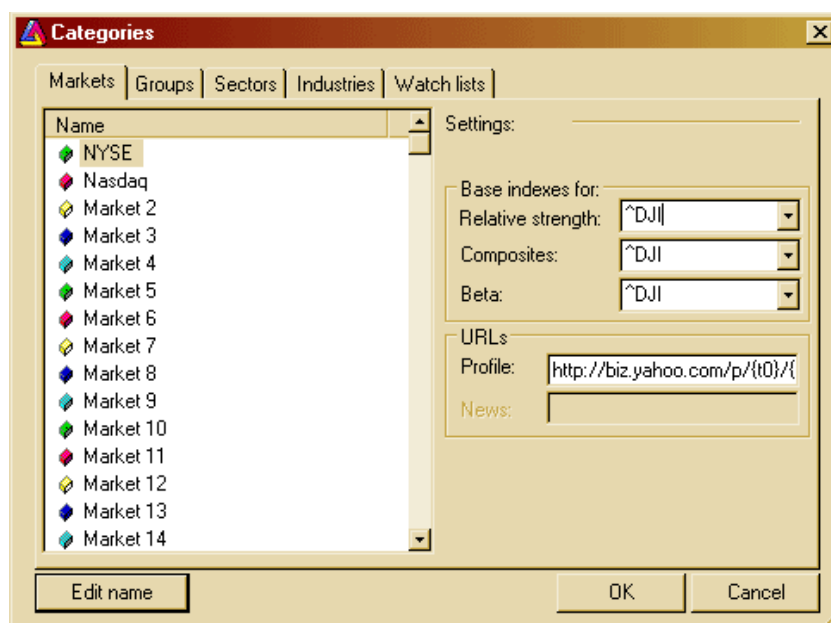
- Open Categories window using Symbol → Categories menu item.
- Select base index for given market in Markets tab and Base indexes for – Composites combo. For example if you are following NYSE this can be ^DJI (Dow Jones Average) ^DJI must be marked as index in Symbol → Information and must belong to the same market.
- Choose "Symbol → Calculate composites" menu item to open the window shown below and mark : *Number of advancing/declining issues* and choose markets that you calculate composites for and the time range.
- Click Calculate.

There are also two additional fields available:

- **Volume for base index**
- **Copy volume to all indexes**

These fields are provided in case you DON'T have real volume data for index quotes. In that case AmiBroker can calculate volume for index as a sum of volumes of all stocks belonging to given market. First option assigns calculated volume only to *base index*, the second copies the volume figure to all indices belonging to given market.

## Categories window



This dialog, allows you to define names of markets, groups, sectors and industries. For each market you can also define base indexes for calculating relative strength, composite data, beta or web profile URL. The detailed information about categories can be found in [Understanding categories](#) chapter of this manual.

To **Edit name** of certain category, please select it from the list and press 'Edit name' button.

**Base indexes for** fields allow you to set the index used in calculation of:

- [Relative Strength](#) indicator
- Composites via [Composite calculation](#) option
- Beta

**Profile** field allows you to define URL-template for viewing on-line (or off-line) companies' profiles. These URL-templates are market-based, what means you can have different templates for each market. The template is then parsed to create the actual URL to the web page, which will be displayed in an embedded web browser. To learn more read [How to set up the profile view](#) chapter.

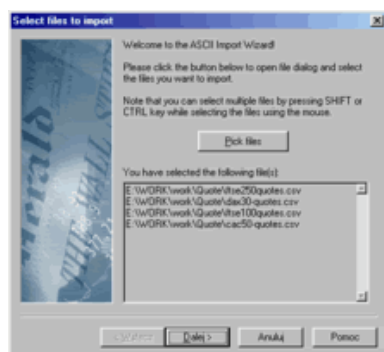
## ASCII Import Wizard

ASCII Import Wizard provides an easy way to import your quotation data files as well as define your own import formats for future use. Note that wizard offers only a subset of features available in [ASCII importer](#) so it is provided for novice users only.

The wizard guides you through 3 simple steps

1. Picking the files to import
2. Defining fields
3. Additional settings

Step 1. Picking the files



In this step you select the files you want to import. Just click on the **Pick files** button and you will see a file dialog. Browse to the folder where your data files are located and select the file(s). Please note that you can select multiple files by holding CTRL or SHIFT key while clicking on the files. After making your selection please click **Open**

A complete list of files that you have selected will be displayed in the field at the bottom of the wizard window. Please check if the list is correct, if not click "Pick files" to correct your choice.

## Step 2. Defining fields



In this step you define the types of fields in the data file. For your convenience date file sample is shown (a few first lines of the first selected file) at the bottom of the window.

To define fields please select appropriate field types from **Column N** combo-boxes. For example, if the first field (column) in your data file is a symbol ticker please select "Ticker" from **Column 1** combo box. If second field in your data file is a date in Year–Month–Day format please select "YMD" from the second combo-box. You can select also DMY for Day–Month–Year dates, MDY for Month–Day–Year dates. Other field types available from the wizard are: "Open", "Close", "High", "Low" for the prices and "Volume".

Note about the dates: AmiBroker recognizes both 4 digit and 2 digit year dates. As for months both numbers and three letter codes ("Jan", "Feb", ...) are allowed. Also day, month and year may be separated by any of the following characters: / (slash), \ (backslash), – (minus sign) or may not be separated at all. All you have to do is to specify the order: DMY, MDY, YMD. For example valid YMD dates are (31th December 2000):

```
20001231,
001231,
2000–12–31
2000/12/31
2000–Dec–31
```

00-12-31

00/12/31

00\12\31

If your file has more than 7 columns please check **More columns** box and you will see additional combo-boxes.

The remaining controls here are:

**Group:** here you should select to which group new symbols are added

**Watch list:** here you should select to which watch list new symbols are added (if empty – they are not added to any watch list)

**Separator:** here you should select the character used as a field separator (comma is the most often)

**Skip lines:** this tells AmiBroker how many initial lines should be skipped (ignored) – for example a few first lines of the file should contain a comment or other information that should be ignored, and this is the place to define this

**Log errors:** this tells AmiBroker that it should log all errors to the file (import.log). In case of any errors this log will be displayed to the user after finishing import process.

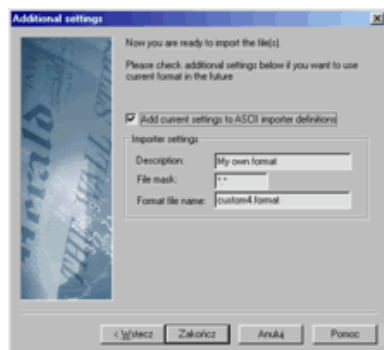
**Automatically add new symbols:** this tells AmiBroker to add the symbols that appear in the data file but do not exist yet in AmiBroker database.

**Calculate composites:** this tells AmiBroker to calculate advance/decline figures and volume for indexes after import (this requires composites to be set up properly before importing)

**Allow negative prices:** this tells AmiBroker to allow negative number in close, open, high, low fields. By default zero and negative values are NOT allowed.

**No quotation data:** allows to import data that do not contain prices. For example ticker lists and/or categories.

### Step 3. Additional settings



By default the format you have defined is for single-use only. It is OK for novice users and for experimenting with the wizard.

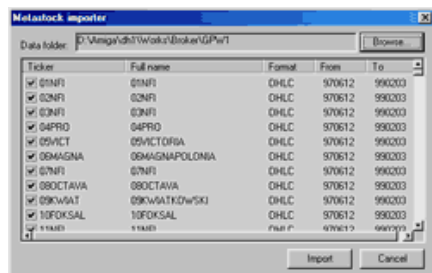
If you, however, want to make your definition permanent and available in the future via [ASCII importer](#) you should check **Add current settings to ASCII importer definitions** box. Then you should enter the **Format description**, **File mask** and **Format file name** (or you can accept automatically generated defaults). If you do so, you will be able to use the format defined in the ASCII importer window – just by selecting your own format (as typed in **Format description** field) from the "Files of type" combo of a file dialog.

Whatever you decide, you should click "Finish" button in order to start importing your data.

## Metastock importer window

**IMPORTANT NOTE:** Metastock importer should be used **ONLY** if you want to import MS data to native, local AmiBroker database once. If you want AmiBroker to just read Metastock database **DIRECTLY** without need to import new data over and over please set up your database [WITH METASTOCK PLUGIN](#) as described in the [Tutorial](#).

**NOTE 2:** if you setup your database with the **MS plugin** you should **NOT** use Metastock importer, because there is no point in using it when your data are already fed by the plugin.



Metastock importer opens AmiBroker to very rich source of historical data. The importer supports both old Metastock 6.5 and new 7.x (XMASTER) formats.

Basically Metastock data consist of:

- MASTER/EMASTER file which holds general information about the tickers, stock names, etc.
- F1.DAT....Fxx.DAT files which hold actual quotation data

The MASTER/EMASTER file is essential because it holds the references to Fxx.DAT files. Fxx.DAT files store only quotations in either 5 field (date/high/low/close/volume), 6 or 7 field (date/open/high/low/close/volume/openinterest) format. As you see MASTER/EMASTER and Fxx.DAT files are closely connected and you need them all to import the data.

## Usage

To import Metastock data you should do the following:

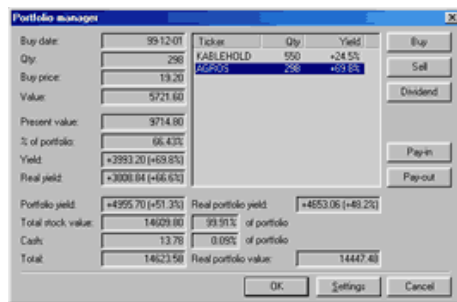
- Choose *Metastock import* from the menu
- Using the directory requester (**Browse...**) select the location of data in Metastock format (the directory with MASTER/EMASTER and Fxx.DAT files)
- After choosing proper directory AmiBroker will display the list of available symbols and date ranges. By default all available symbol will be marked for importing (checkmark at the beginning of the list). Now you can exclude some symbol from the import list by clicking appropriate item in the list (checkmark will toggle when you click).



- You can decide to which group and watch list the new symbols are added using **Group** and **Watch List** combos.
- After making your selections push '**Import**' button to start the process of importation.
- During the process you can cancel the operation by clicking '**Abort**' button in the progress window

## Portfolio management window

**NOTE:** This document describes functionality that is **PHASED OUT** and replaced in version 4.90 by superior **Account Manager**.



This window lets you manage your portfolio of symbols. You can perform purchase and sell operations and the program will calculate commissions for you.

There are following items in the portfolio management window:

- portfolio symbol list
- text fields containing date, quantity, buy price, current price, value, yield, and 'real' yield (this is yield minus commission)
- total security value
- cash remaining
- total portfolio value

Following actions can be taken:

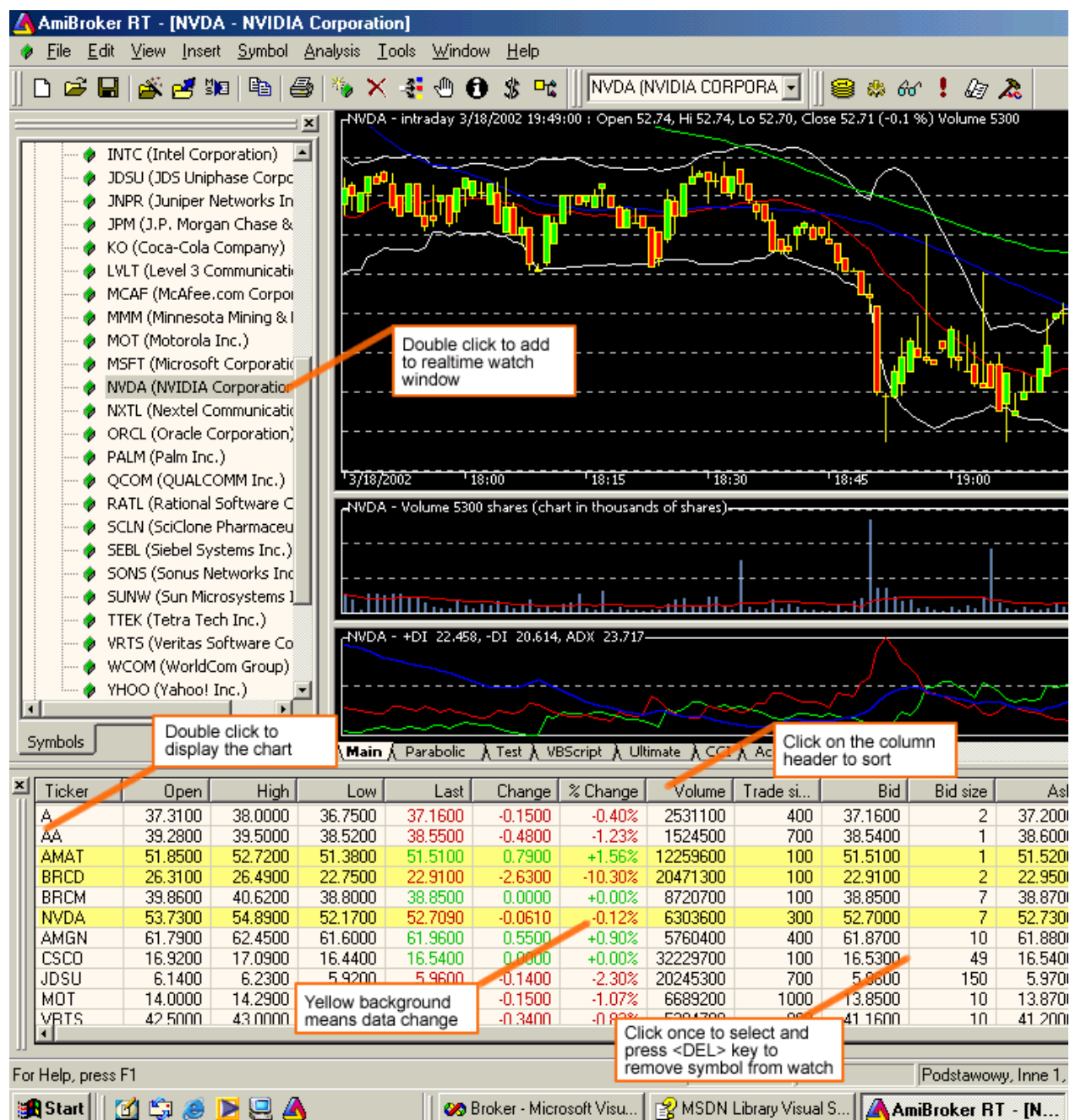
- Buy securities  
you will be asked for confirming symbol selection and price and then you will be asked for a number of shares you want to buy
- Sell securities  
as before except of the fact that you have to choose the symbol from the portfolio symbol list.
- Dividend  
after selecting that control you will be asked for dividend value per one share. Program will calculate total dividend value minus income tax.
- Deposit  
after pressing this button you will be asked for amount of cash you want to deposit.
- Withdraw  
after pressing this button you will be asked for amount of cash you want to withdraw

The commissions taken by all buy/sell transactions in are definable in *Settings (Commission)* window.

## Real-time quote window

### Working with real time quote window

The RT quote window provides real-time streaming quotes and some basic fundamental data. It is fairly easy to operate as shown in the picture below:



You can also display context menu by pressing RIGHT mouse button over RT quote window.

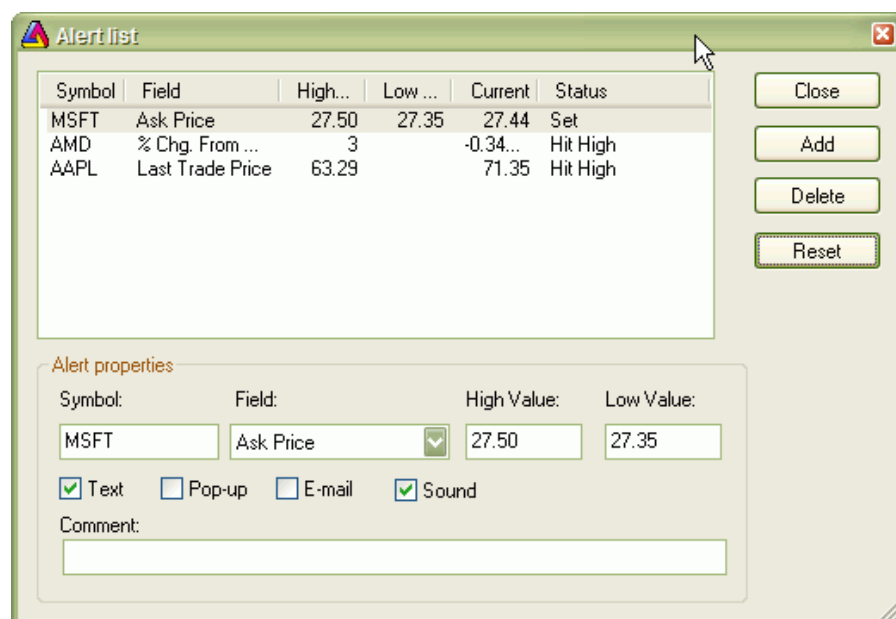
Realtime Quote							
Ticker	Open	High	Low	Last	Change	% Change	Volum
AAPL	68.2950	71.4300	68.2000	71.1700	3.9600	+5.89%	60,459,82
ADBE	35.9400	36.1000	35.6000	36.0000	0.0100	+0.03%	1,860,12
ALTR	20.8200	21.2600	20.7500	20.9100	0.0400	+0.19%	2,722,91
AMAT	18.3200	18.6000	18.2600	18.4700	0.2100	+1.15%	12,447,14
AMGN	72.5600	73.1200	71.5700	71.7200	-1.1400	-1.56%	5,411,84
AMZN	37.2000	38.4300	37.1300	38.3500	1.0100	+2.70%	4,251,95
APCC	23.0200			23.0100	0.0300	+0.13%	588,46
APOL	52.9500			52.7100	1.5300	+2.99%	2,546,04
ATYT	16.9400			17.0800	0.1400	+0.83%	5,244,00
BBBY	40.8400			40.7400	2.4200	+6.32%	11,001,66
BEAS	13.3800			12.9800	-0.4200	-3.13%	5,923,80
BIBB	45.3000			44.7775	-0.5825	-1.28%	1,382,83
BMET	37.7500			38.0000	0.2200	+0.58%	1,622,57
BRCM	45.8500			46.2300	0.1760	+0.38%	8,992,08
CDWC	58.7500			58.0630	-0.7070	-1.20%	362,83
CECO	40.8300			40.7200	-0.9800	-2.35%	1,708,99

The context menu allows you to access the following options:

- **Time & Sales**  
Opens [Time & Sales window](#) that provides information about every bid, ask and trade streaming from the market.
- **Easy Alerts**  
Opens [Easy Alerts window](#) that provides way to define realtime alerts executed when bid/ask/last and other fields hit user-defined levels
- **Add Symbol**  
Adds current symbol to Real-Time Quote list
- **Add watch list...**  
Adds entire watch list to real-time quote window
- **Remove Symbol**  
Removes highlighted line (symbol) from the Real-Time Quote list.
- **Remove All**  
Removes all symbols from real-time quote list
- **Hide**  
Hides Real-Time Quote list

## Easy alerts window

Easy alert window allows to define real-time alerts without any coding. Please note that this functionality is available ONLY if you are using real-time data plugin and is not available in end-of-day mode.



### Adding new alert

- press **Add** button
- enter ticker symbol into **Symbol** field
- choose price field from **Field** combo box
- enter high trigger value and/or low trigger value
- select at least one of the **Text/Pop-up/E-mail/Sound** fields

Alert will be generated when selected price field (for example Ask) will become equal or greater than High value (if defined), or when selected price field will become equal or less than Low value (if defined). Alert once hit will not re-trigger until you press "**Reset**".

### Modifying an alert

Select one of listed alerts and modify values in the edit fields below. If you want to modify an alert that was hit already, after doing modifications please press "**Reset**" button

### Deleting alerts

Select one or more alerts from the list (multiple selection possible by pressing down SHIFT key) and then press **Delete** button.

### Resetting triggered alerts

The alert that was once hit is marked as "Hit high" or "Hit low" in the status field and becomes inactive (won't trigger anymore). If you want to re-activate it, select it from the list and press **Reset** button.

### Kinds of alert output

- **Text**  
when this checkbox is marked, when alert is triggered the text defined in comment field will be displayed in Alert Output window (use View->Alert output menu to display it)

- **Pop-up**  
when this checkbox is marked, triggered alert will display pop-up dialog box
- **E-mail**  
when this checkbox is marked, triggered alert will send an e-mail to the account defined in [Preferences/Alerts](#).
- **Sound**  
when this checkbox is marked, triggered alert will play sound defined in [Preferences/Alerts](#).

#### Time & Sales window

*NOTE: Standard Edition is limited to 1 time & sales window, Professional Edition allows UNLIMITED number of time & sales window open simultaneously.*

Time & Sales window that provides information about every bid, ask and trade streaming from the market. Each row displayed represents either new trade, new bid or new ask that is sent by the streaming data source.

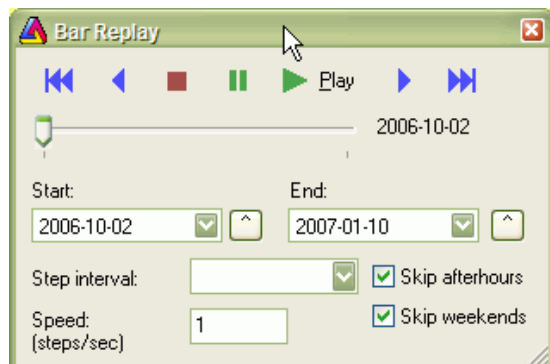
Each line in time and sales window is marked with color to make it easier to distinguish between various conditions.

Coloring rules are:

- light green background means NEW ASK
- light red background means NEW BID
- normal (white) background means NEW TRADE
- Red text for bid/ask price/size means that the value is LESS than previous value of the same category (for example bid price written in red letters mean that the new BID is lower than previous bid price, green volume field means that the volume of last trade or ask/bid size is greater than last trade volume or ask/bid size)
- Red last trade price means trade occurring on or below current bid
- Green text for bid/ask price/size means that the value is GREATER than previous value.
- Green last trade price means trade occurring on or above current ask.
- Black text for bid/ask price/size/volume means that the value is the same
- Black last trade price means trade occurring inside current bid-ask range (greater than bid and less than ask)









## Bar Replay window

Bar Replay tool is available from **Tools→Bar Replay** menu. Bar Replay feature **plays back data for all symbols at once** with user-defined speed. It means that data for all symbols will end at specified "playback position". This affects all formulas (no matter if they are used in charts (indicators) or auto-analysis).



### Controls description

Navigation bar:

-  – **Rewind** to the beginning
-  – one **Step Back**
-  – **Stop** – turns bar replay OFF (chart are not affected by bar reply)
-  – **Pause** – pauses current playback or enters pause mode that allows to manually drag the slider bar and affect chart display – in PAUSE mode data are internally modified so quotes past selected "playback" position are invisible to any part of AmiBroker ( charts / automatic analysis ), except quotation editor
-   – **Play** – playback bars history
-  – one **Step Forward**
-  – **Forward** to end of selected range

**Slider bar** – allows to see the playback progress as well as MANUALLY move back and forward by dragging the slider thumb.

**Start/End** – controls provide start and end simulation dates. The playback works so that all data upto currently selected "Playback position" are visible. Data past this position are invisible. "Playback position" can change from user-defined "Start" to "End" dates. The small ^ buttons on the right side of Start / End date fields allow to set Start/End to currently selected date on the chart.



**Step interval** – defines interval of the step. Recommended setting is base interval of your database. So if you have 1-minute database, step interval should be 1 minute. If you have EOD database, step interval should be daily, however it is allowed to select higher step intervals. Note that chart viewing interval is independent from that. So you can playback 1 minute database and watch 15 minute bars (they will look like real – building last "ghost" bar as new data come in)


**Speed** parameter defines step frequency. It means how many steps will be played back within one second. Default is 1. Maximum is 5 minimum is 0.1. If you select 3 for example, AmiBroker will play one step every 0.333 sec giving total of 3 steps per second.

**Skip afterhours** – when turned on, playback skips hours outside regular trading hours as defined in File→Database Settings→Intraday Settings

**Skip weekends** – when turned on, playback skips Saturdays and Sundays

## Usage

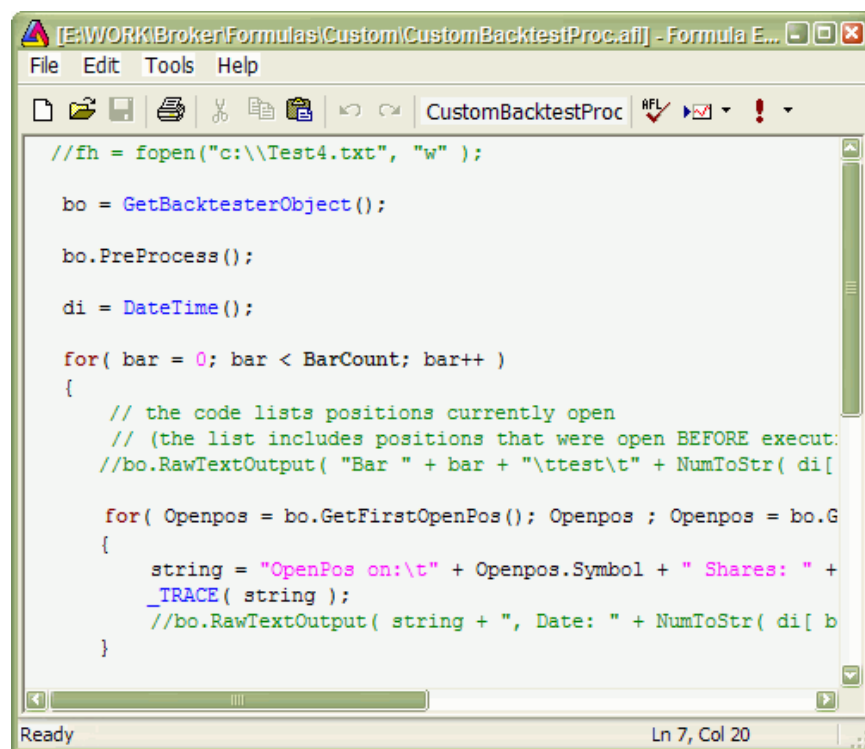
To ENTER Playback mode – press PLAY  or PAUSE  buttons – then data are truncated at current "playback position".

To EXIT Playback mode – press STOP  button or close Bar Replay dialog – the full data set will be restored.

Note that playback simulation is done internally and the database is kept untouched in fact (all data are still visible in Quote Editor), so there is no risk using Bar Reply.

## Formula Editor

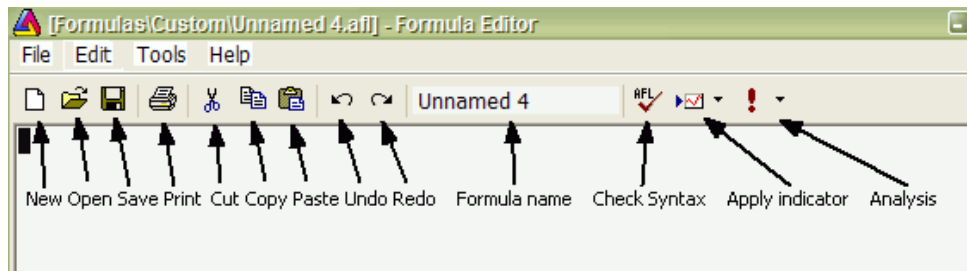
AFL Formula Editor features user-definable syntax highlighting, context-sensitive formula reference help, enhanced error reporting, automatic statement completion and parameter information technology (similar to Intellisense(tm) featured in Microsoft Developer studio), support for editing multiple files at once, and is multi-monitor friendly. These features greatly simplifies writing formula and provides instant help so time needed to write formula decreases significantly.



## Menu

Formula Editor menu options are described in detail in [Menus: Formula Editor](#) chapter of the guide.

## Toolbar



The Formula Editor toolbar provides the following buttons:

- **New** – clears the formula editor window
- **Open** – opens the formula file
- **Save** – saves the formula under current name
- **Print** – prints the formula
- **Cut** – cuts the selection and copies to the clipboard
- **Copy** – copies the selection to the clipboard
- **Paste** – pastes current clipboard content in the current cursor position
- **Undo** – un–does recent action (multiple–level)
- **Redo** – re–does recent action (multiple–level)
- **Formula Name** – an EDIT field that allows to modify the formula file name, once you change the name here and press **Save** button the formula will be saved under new name and the change will be reflected in editor CAPTION BAR.
- **Check syntax** – checks current formula for errors
- **Apply indicator** – saves the formula and applies current formula as a chart/indicator ONCE
- **Analysis** – saves the formula and selects it as current formula in Automatic Analysis window and repeat most recently used Analysis operation (i.e. Scan or Exploration or Backtest or Optimization)

## Usage

Typical use of Formula Editor is as follows:

- open Formula Editor
- type the formula
- type meaningful name that describes the purpose of you code into **Formula Name** field
- click **Apply indicator** button (if you have written indicator code)  
.. or..  
click **Analysis** button to display Automatic Analysis window (when you have written exploration/scan or trading system)

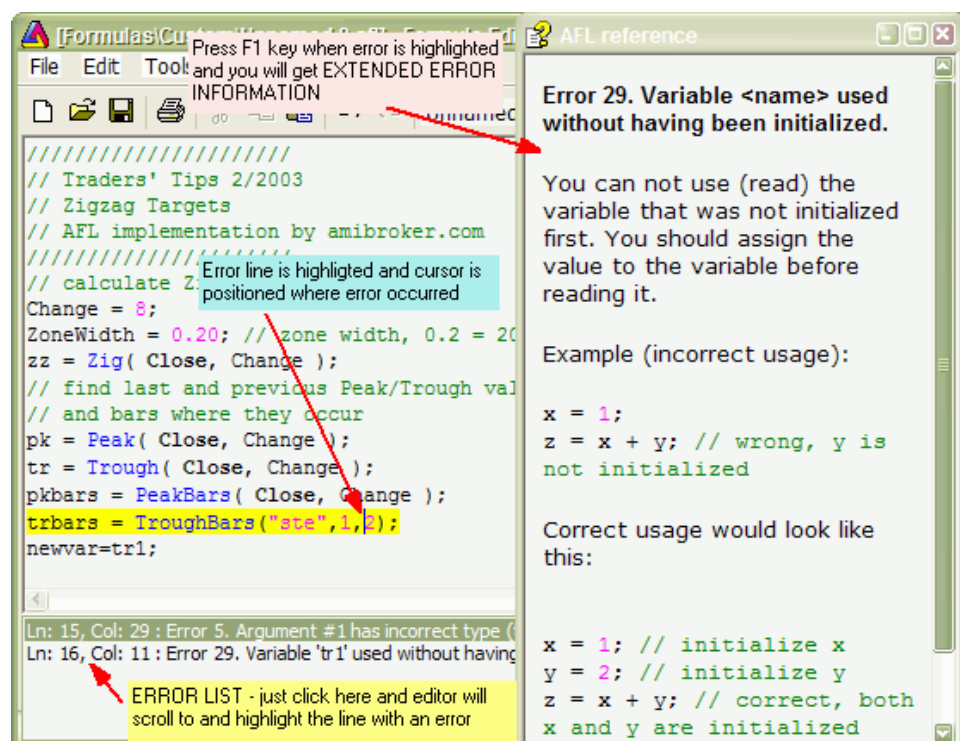
## Syntax highlighting

AmiBroker's AFL editor features user–definable syntax highlighting that automatically applies user–defined colors and styles to different language elements like functions and reserved variable names, strings, numbers, comments, etc. This feature greatly simplifies code writing. You can modify coloring scheme in [Preferences window](#).

## Enhanced error reporting

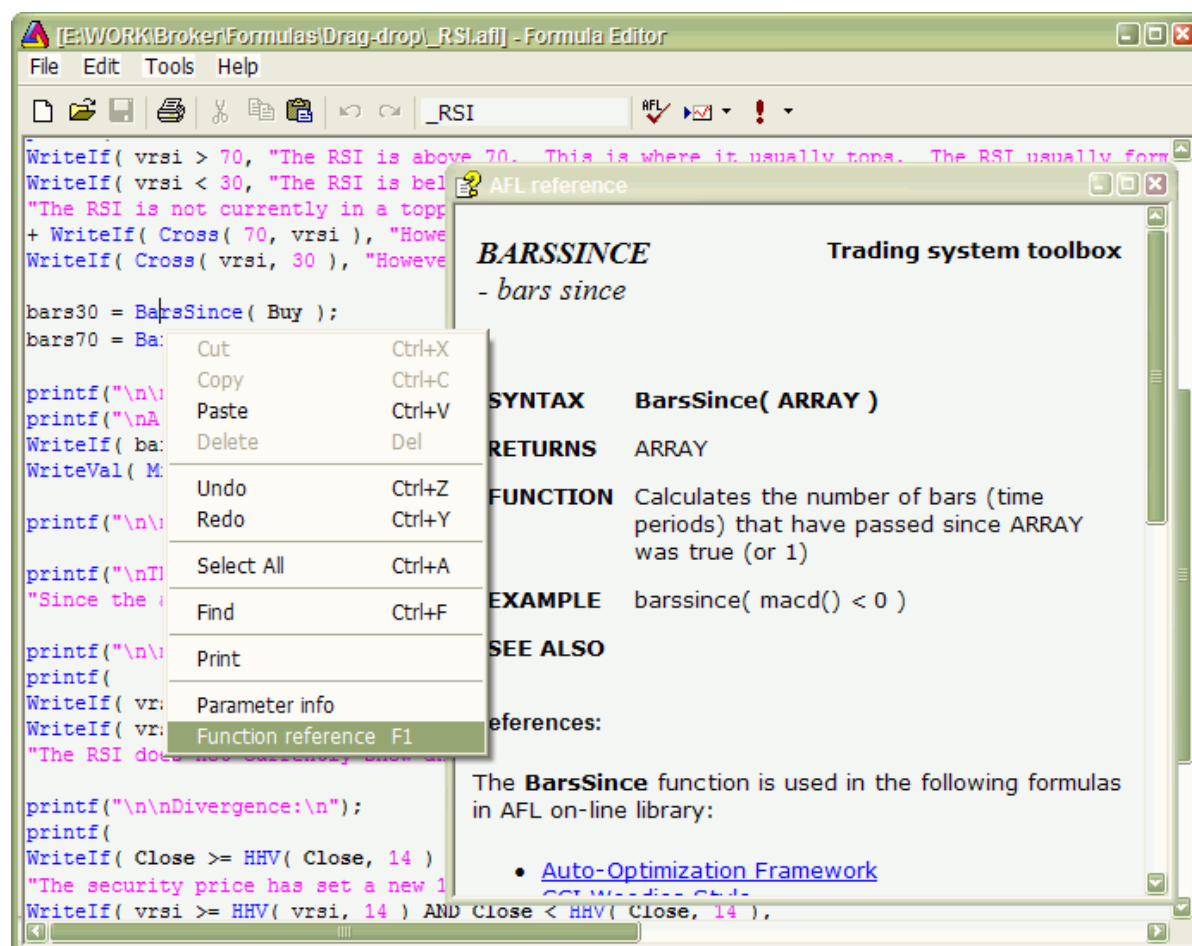
When you make an error in your formula, AmiBroker's enhanced error reporting will help you to locate and fix an error by highlighting the place where error occurred and displaying extended error description with the examples of common mistakes and advice how to fix them.





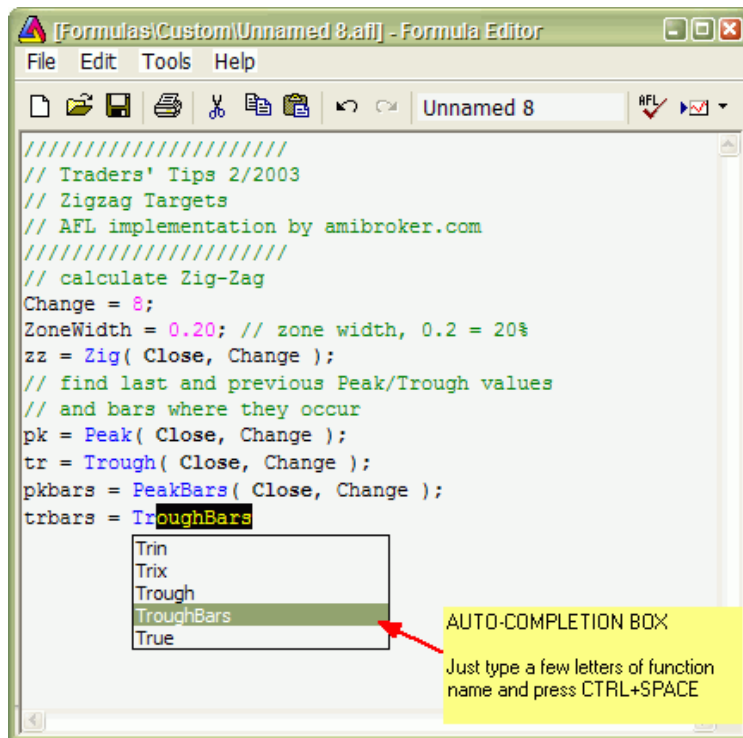
### Context help

You can quickly display relevant AFL function reference page if you press **F1** key or choose "Function reference" from the context menu while the caret is inside or right after function name as shown in the picture below:



### Automatic statement completion

The automatic completion feature (available when you press **CTRL+SPACE** key combination) finishes typing your functions and reserved variables for you, or displays a list of candidates if what you've typed has more than one possible match. You can select the item from the list using up/down arrow keys or your mouse. To accept selection press RETURN (ENTER). You can also type immediately space (for variables) or opening brace (for function) and AmiBroker will auto-complete currently selected word and close the list. To dismiss the list press ESC key.



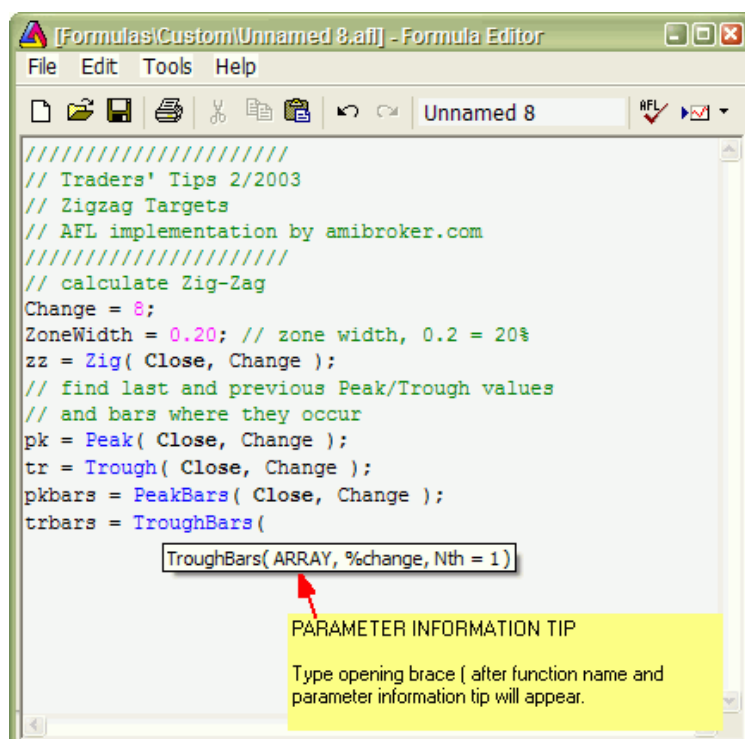
### Parameter Information

When you are typing a function, you can display a Tool Tip containing the complete function prototype, including parameters. The **Parameter Info** Tool Tip is also displayed for nested functions. With your insertion point next to a function, type an open parenthesis as you normally would to enclose the parameter list.

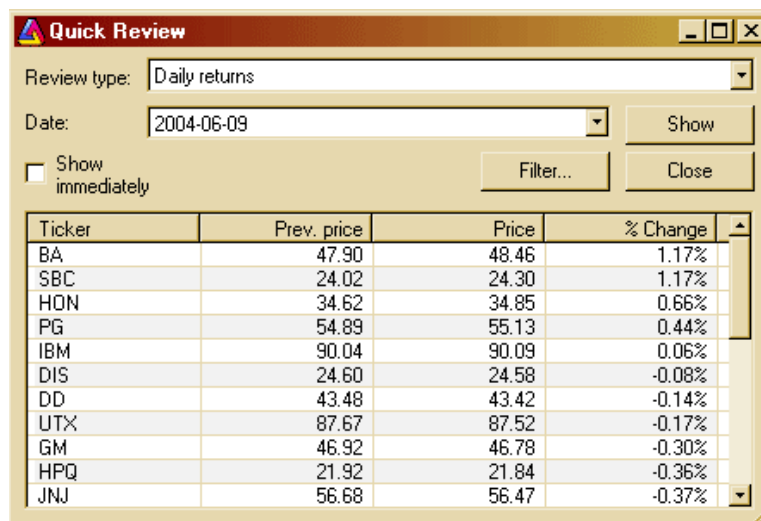
AmiBroker displays the complete declaration for the function in a pop-up window just under the insertion point.

Typing the closing parenthesis dismisses the parameter list.

You can also dismiss the list if you press arrow up/down key, click with the mouse or press RETURN.



## Quick review window



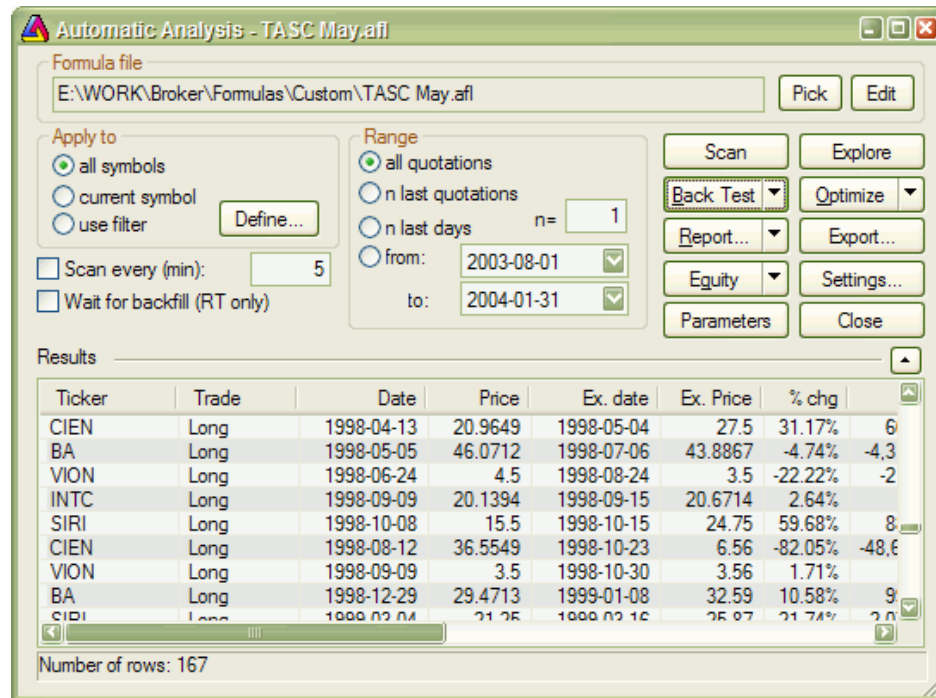
This window provides overall market information like:

- daily symbol quotes
- weekly returns comparison table
- monthly returns comparison table
- quarterly returns comparison table
- yearly returns comparison table
- Price/Earnings comparison
- Price/Book value comparison

In the **Date** field you select the base date for comparisons. For example weekly returns are calculated by dividing base day close price with the closing price one week before.

**Filter** button allows you to narrow down your search to symbols defined in [Filter settings window](#).

## Automatic analysis window



Automatic analysis window enables you to check your quotations against defined buy/sell rules. AmiBroker can produce report telling you if buy/sell signals occurred on given symbol in the specified period of time. It can also simulate trading, giving you an idea about performance of your system.

In the upper part of window you can see the path to the formula used along with **Pick** and **Edit** buttons.

**Pick** button opens up a file dialog that allows you to choose the formula you want to use for the analysis.

**Edit** button opens up the [AFL Formula Editor](#) that allows you to edit currently selected formula.

If you want to create new formula just open [Formula Editor](#) directly from **Tools→Formula Editor** menu, type the formula and press **Analysis** button in the Formula Editor toolbar.

In the formula editor you need to write the code that specifies either [scan/exploration](#) you want to run or a [trading system](#) you want to back test. You can find the description of this language in [AFL reference guide](#).

In order to make things work you should write two assignment statements (one for buy rule, second for the sell rule), for example:

```
buy = cross( macd(), 0 );
sell = cross( 0, macd() );
```

Below these fields there are several controls for setting:

1. To which symbol(s) analysis should be applied.  
You can select here all symbols, only currently selected symbol (selected in Select Window) or [custom filter](#) setting
2. Time range of analysis  
analysis can be applied to all available quotations or only to the defined number of most recent quotations (or days) or to a date range (from/to)

In the lower part of the analysis window you can see 4 buttons:

1. **Scan**  
this starts the signal scan mode – AmiBroker will search through defined range of symbols and quotations for buy/sell signals defined by your trading rules. If one of the buy/sell conditions is fulfilled, AmiBroker will display a line describing when and on which symbol the signal has occurred. Next AmiBroker proceeds to the end of the range so multiple signals on single symbol may be generated.
2. **Explore**  
this starts an exploration mode when AmiBroker scans through database to find symbols that match user-defined filter. The user can define output columns that show any kind of information required. For more information please check out "[Tutorial: How to create your own exploration](#)"
3. **Back Test**  
this starts the back-testing mode – AmiBroker will search through defined range of symbols and quotation for BUY signal defined by your buy rule. If the buy rule is fulfilled, AmiBroker will "buy" currently analyzed shares. Next it will search for SELL signal. Then, if sell rule is fulfilled, AmiBroker will "sell" previously bought symbols. At the same time it will display the information about this trading in the listview. After performing simulation the summary will be displayed. [Read more in "Tutorial: How to backtest your trading system"...](#)  
The back testing parameters could be changed using [Settings](#) window.
4. [Settings](#) – allows you to change back tester settings
5. **Optimize** – allows you to optimize your trading system. [Read more in the "Tutorial: How to optimize your trading system"...](#)
6. **Check** – this option allows you to check if your formula references future quotes. AmiBroker analyses your formula and detects if it uses quotes past current bar. Please note that formulas referencing future can give unrealistic backtesting results that can not be reproduced in real trading, therefore you should avoid systems that reference future.
7. **Report**  
this displays [Report window](#) that allows you to watch, print and save test results
8. **Equity**  
– available only after backtesting – displays Equity curve for currently selected symbol in a new chart pane. Check out "[AFL: Equity chart and function](#)".
9. **Export** – allows you to export the results to CSV (comma separated values) file
10. **Close**  
this closes the analysis window

Moreover you two options "Load" and "Save" for loading and saving your trading rules from/to files.

### Enlarging results view in Automatic analysis window

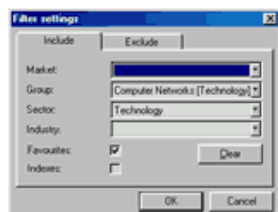
There is a small arrow button next to the "Result list" horizontal divider line. This button is provided to enlarge/shrink the result list. When you are editing your formula it is good to have edit view larger, but to see the backtesting results it is convenient to enlarge the result list. In that case just click on that button and the result list will be enlarged (and the edit field will get shrunk). To do the reverse just click the button again.

## Filter settings window

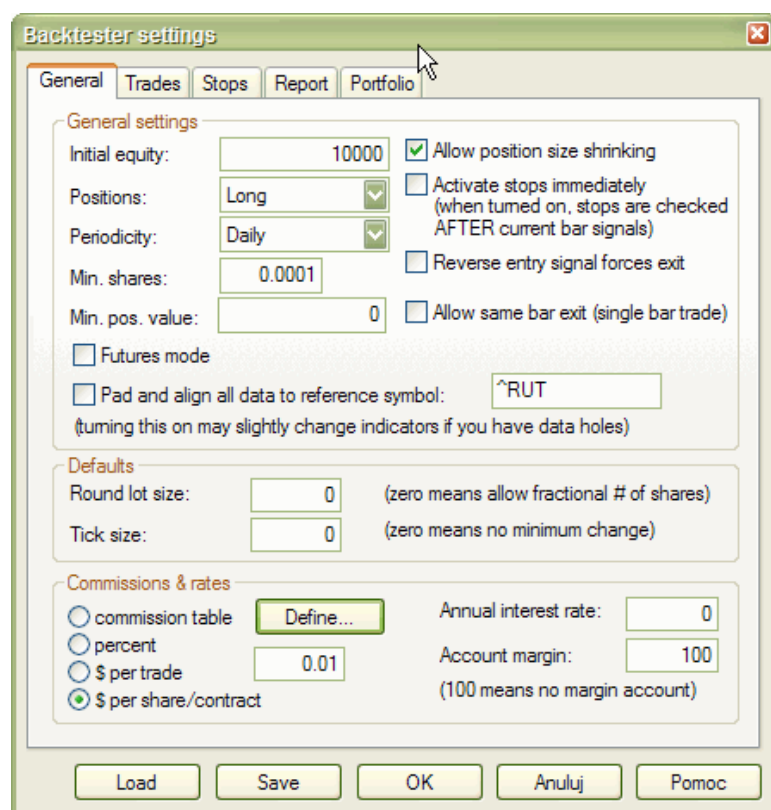
This window is available from "Filter/Define..." button in quick-review and analysis windows.

Filtering option gives you ability to narrow your search to symbols belonging to the specified market, group, sector and industry. You can also mark to include only favourites or indexes. You can use include and/or exclude type filter so you can also selectively exclude some kind of symbols .

If you use more than one category (for example you select Market and Sector ) the filter will pass only those symbols that match first AND second category (this logical conjunction, not alternative)



## System test settings window



Here you can define the following parameters of back-testing:

### General tab

*Initial equity* – defines the size of your account. In Portfolio backtest – it represents entire portfolio size. In "Individual" backtest it is per-symbol initial equity.

*Positions considered (long, short, both long and short)*

*Futures mode*

This check box in the settings page is the key to backtesting futures. It instructs backtester to use margin deposit and point value in calculations.

*Min. shares*

The minimum number of shares that are allowed to buy/short. Backtester will not enter trades below that limit. Should be 1 for stocks. Fractional values are good for mutual funds.

*Min. pos value*

The minimum position value (in base currency) of the trade that is allowed to be entered. Backtester will not enter trades below that limit. Zero means no limit.

*Pad and align to reference symbol*

When this is turned on, all symbols' quotes are padded and aligned to reference symbol. Note: by default this setting is OFF. Use responsibly. It may slow down backtest/exploration/scan and introduce some slight changes to indicator values when your data has holes and holes are filled with previous bar data. The feature is intended to be used when your system uses general market timing (generates global signals based on data and/or indicators calculated using Foreign from 'reference' symbol) or when you are creating composites out of unaligned data. Note: if reference symbol does not exist, data won't be padded.

*Account margin*

This setting defines percentage margin requirement for entire account. The default value of Account margin is 100. This means that you have to provide 100% funds to enter the trade, and this is the way how backtester worked in previous versions. But now you can simulate a margin account. When you buy on margin you are simply borrowing money from your broker to buy stock. With current regulations you can put up 50% of the purchase price of the stock you wish to buy and borrow the other half from your broker. To simulate this just enter 50 in the Account margin field (see pic. 1) . If your initial equity is set to 10000 your buying power will be then 20000 and you will be able to enter bigger positions. Please note that this settings sets the margin for entire account and it is NOT related to futures trading at all. In other words you can trade stocks on margin account.

*Commissions*

- **commission table** – backtester will use commission table as defined in [Commission Schedule table](#) window (press **Define...** button to show it).
- **percent** – commission is expressed as a percent of trade value
- **\$ per trade** – commission is fixed amount of dollars (or your currency) per trade
- **\$ per share/contract** – commission is expressed in dollars (or your currency) per share/contract purchased/sold

*Annual interest rate*

This setting allows you to define annual interest earned when you are out of the market or your position is less than available equity.

*Filter settings window*



### *Periodicity*

This setting controls bar interval used for backtesting/scan/exploration/optimization. To backtest intraday data you should switch to proper interval there and then run the backtest.

### *Allow position size shrinking*

If you mark this box AmiBroker will shrink down positions if available equity is less than requested position size (via PositionSize variable). If this box is unmarked positions will not be entered in such case.

### *Activate stops immediately*

When you trade on open and want to have built-in stops activated on the same bar – just mark this box.

If you trade on close and want built-in stops to be activated from the next bar – unmark this box.

You may ask why do not simply check the buyprice or shortprice array if it is equal to open price. Unfortunately this won't work. Why? Simply because there are doji days when open price equals close and then backtester will never know if trade was entered at market open or close.

### *Round lot size*

Various instruments are traded with various "trading units" or "blocks". For example you can purchase fractional number of units of mutual fund, but you can not purchase fractional number of shares. Sometimes you have to buy in 10s or 100s lots. AmiBroker now allows you to specify the block size on global and per-symbol level.

You can define per-symbol round lot size in the Symbol->Information page. The value of zero means that the symbol has no special round lot size and will use "Default round lot size" (global setting) from the Automatic Analysis settings page. If default size is set also to zero it means that fractional number of shares/contracts are allowed.

You can also control round lot size directly from your AFL formula using RoundLotSize reserved variable, for example:

```
RoundLotSize = 10;
```

### *Tick size*

This setting controls the minimum price move of given symbol. You can define it on global and per-symbol level. As with round lot size, you can define per-symbol tick size in the Symbol->Information page. The value of zero instructs AmiBroker to use "default tick size" defined in the Settings page of Automatic Analysis window. If default tick size is also set to zero it means that there is no minimum price move.

You can set and retrieve the tick size also from AFL formula using TickSize reserved variable, for example:

```
TickSize = 0.01;
```

Note that the tick size setting affects ONLY trades exited by built-in stops and/or ApplyStop(). The backtester assumes that price data follow tick size requirements and it does not change price arrays supplied by the user.

So specifying tick size makes sense only if you are using built-in stops so exit points are generated at "allowed" price levels instead of calculated ones. For example in Japan – you can not have fractional parts of yen so you should define global ticksize to 1, so built-in stops exit trades at integer levels.

#### *Reverse entry signal forces exit*

When it is ON (the default setting) – backtester works as in previous versions and closes already open position if new entry signal in reverse direction is encountered. If this switch is OFF – even if reverse signal occurs backtester maintains currently open trade and does not close position until regular exit (sell or cover) signal is generated.

In other words when this switch is OFF backtester ignores Short signals during long trades and ignores Buy signals during short trades.

#### *Allow same bar exit (single bar trade)*

When it is ON (the default settings) – entry and exit at the very same bar is allowed (as in previous versions) if it is OFF – exit can happen starting from next bar only (this applies to regular signals, there is a separate setting for ApplyStop-generated exits). Switching it to OFF allows to reproduce the behaviour of MS backtester that is not able to handle same day exits.

### **Trades tab**

- **prices** buy/sell/short/cover price fields – allows the user to define at which price to buy/sell/short sell/buy to cover during system test
- **delays** buy/sell/short/cover delay – allows to define custom delay between signal and trade

### **Stop tab**

- max. loss stop
- profit target stop
- trailing stop
- N-bar stop

See [APPLYSTOP](#) function for more details on different stop settings

### **Report tab**

#### *Result list shows*

This decides which format of result list is used by new backtester. Possible choices:

- Trade list (the default) – each trade is listed in a separate row. Trades are ordered by exit date by default
- Detailed log – each data bar is listed separately. The log shows scores, positions and other very detailed information useful for debugging your trading system/position sizing/scoring strategies
- Summary – one row per backtest is generated. The row contains backtest summary/statistics (like the report)

#### *Risk free rates*

Defines risk free rates for Sharpe and UPI stats

### *Filter settings window*

*Distribution charts spacing*

Defines the spacing of profit, MAE and MFE distribution charts. The spacing is the % amount of profit/MAE/MFE per single bar in a chart.

*Generate detailed reports for individual backtests*

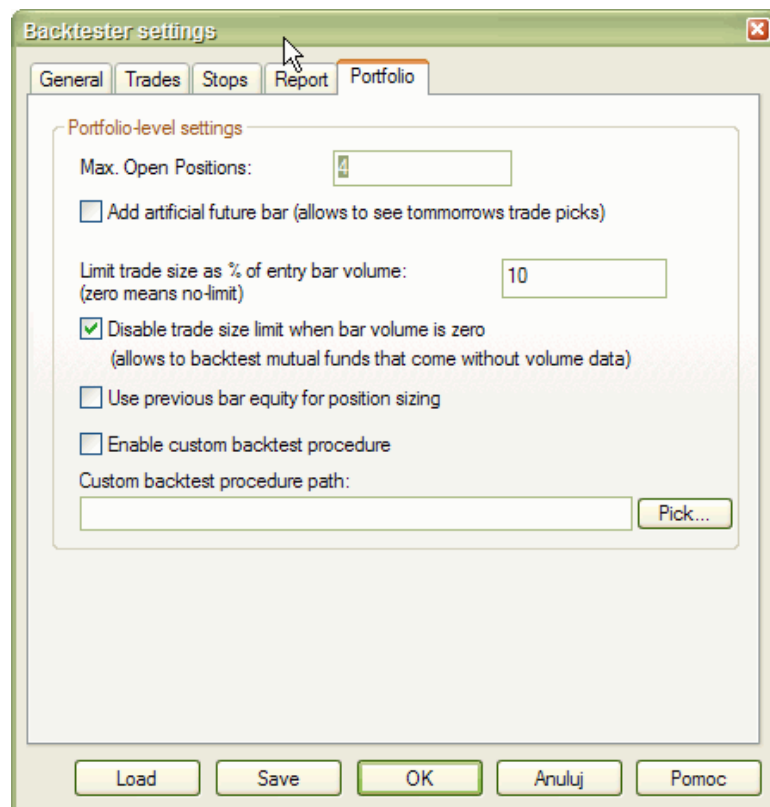
This causes that in Individual backtest mode full report is generated and stored for every security under test. Note that this will slow down the test and take up quite a bit of hard disk space

*Include trade list in the report*

When turned ON (by default) the backtest report includes also trade list. Note that trade lists may be huge and consume quite a bit of disk space

*Warn before time-consuming optimizations*

When turned ON (by default), AmiBroker will display confirmation dialog box when your optimization has more than 300 steps.

**Portfolio tab***Max. Open Positions*

Max. Open Positions – the maximum number of simultaneously open positions. .Settable also using SetOption("MaxOpenPositions", number ) function.

*Add artificial future bar*

When checked AmiBroker adds tomorrow's bar and this enables you to see tomorrow's (or next bar) trade recommendations when your system uses one bar delay. Artificial future bar has incremented date and volume set to zero and all price fields (OHLC) set to CLOSE price of last data bar.

*Limit trade size as % of entry bar volume*

This prevents from entering the trades greater than given percentage of entry bar's volume. For example if backtesting daily data and today's volume for thinly traded stock is 177,000 shares, setting this to 10% will limit the maximum trade size to 17,700 shares (10% of total daily volume). This prevents from 'affecting the market' by huge orders.

**IMPORTANT NOTE:**

Some instruments like MUTUAL FUNDS come without VOLUME data. To backtest such instruments please set this field to ZERO (0) or check "Disable trade size limit when bar volume is zero" box. This effectively turns OFF this feature. Otherwise you won't be able to enter any trade at all.

*Disable trade size limit when bar volume is zero*

When it is turned ON and the entry bar volume is zero the backtester will not apply the "limit trade size as % of entry bar volume" – this is to allow backtesting mutual funds that come with zero volume data. When it is OFF and entry bar volume is zero then backtester will not allow to enter the trade on such bar.

*Use previous bar equity for position sizing*

Affects how percent of current equity position sizing is performed.

Unchecked (default value) means: use current (intraday) equity to perform position sizing, checked means: use previous bar closing equity to perform position sizing.

*Enable custom backtest procedure*

When checked AmiBroker applies the custom backtest formula specified in the field below to every backtest that you run. This is useful if you want to permanently add your [custom metrics](#) to all backtests without need to copy paste the same code.

*Custom backtest procedure path*

The full path to custom backtest formula (see above).

**Old tab***Drawdown figures based on...*

Drawdown figures in the backtest report measure equity dip experienced during the trade(s). To calculate the dip you can use the worst case scenario: low price for long trades and high price for short trades or single price (open or close) for both long and short trades. "Drawdown figures based on..." setting (pic. 2) allows you to choose the price(s) used to calculate drawdowns. Using worst case scenario you will get a few percent bigger drawdowns than using close or open price. On the other hand Equity() function always uses shortprice/coverprice array so you may choose open or close field here to match drawdowns as observed in equity line.

*Formula*

- mark this box to include AFL formula in the backtest report

*Settings*

- mark this box to include settings in the backtest report

*Incl. out-of-market pos*

- mark this box to include out-of-market positions in the backtest report

*Overall summary*

- mark this box to include sum of individual symbol backtest results

*Symbol summary*

- mark this box to include per-symbol summaries

*Trade list*

- choose format of trade list included in the report

**System test report window****NEW BACKTESTER REPORT**

**Exposure %** – 'Market exposure of the trading system calculated on bar by bar basis. Sum of bar exposures divided by number of bars. Single bar exposure is the value of open positions divided by portfolio equity.

**Net Risk Adjusted Return %** – Net profit % divided by Exposure %

**Annual Return %** – Compounded Annual Return % (CAR) – this is

**Risk Adjusted Return %** – Annual return % divided by Exposure %

**Avg. Profit/Loss** – (Profit of winners + Loss of losers)/(number of trades)

**Avg. Profit/Loss %** – '(% Profit of winners + % Loss of losers)/(number of trades)

**Avg. Bars Held** – sum of bars in trades / number of trades

**Max. trade drawdown** – The largest peak to valley decline experienced in any single trade

**Max. trade % drawdown** – The largest peak to valley percentage decline experienced in any single trade

**Max. system drawdown** – The largest peak to valley decline experienced in portfolio equity

**Max. system % drawdown** – The largest peak to valley percentage decline experienced in portfolio equity

**Recovery Factor** – Net profit divided by Max. system drawdown

**CAR/MaxDD** – Compound Annual % Return divided by Max. system % drawdown

**RAR/MaxDD** – Risk Adjusted Return divided by Max. system % drawdown

**Profit Factor** – Profit of winners divided by loss of losers

**Payoff Ratio** – Ratio average win / average loss

**Standard Error** – Standard error measures chopiness of equity line. The lower the better.

**Risk–Reward Ratio** – Measure of the relation between the risk inherent in a trading the system compared to its potential gain. Higher is better. Calculated as slope of equity line (expected annual return) divided by its standard error.

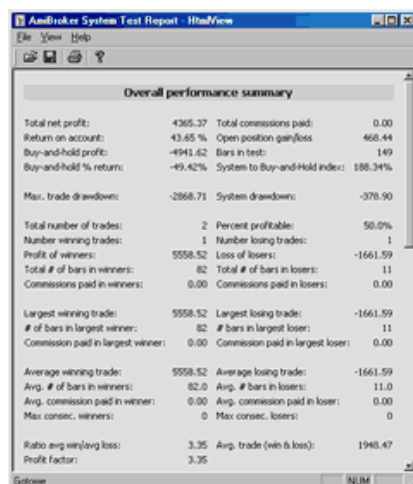
**Ulcer Index** – Square root of sum of squared drawdowns divided by number of bars

**Ulcer Performance Index** – (Annual profit – Treasury notes profit)/Ulcer Index>Ulcer Performance Index. Currently treasury notes profit is hardcoded at 5.4. In future version there will be user–setting for this.

**Sharpe Ratio of trades** – Measure of risk adjusted return of investment. Above 1.0 is good, more than 2.0 is very good. More information <http://www.stanford.edu/~wfs Sharpe/art/sr/sr.htm> . Calculation: first average percentage return and standard deviation of returns is calculated. Then these two figures are annualized by multiplying them by ratio (NumberOfBarsPerYear)/(AvgNumberOfBarsPerTrade). Then the risk free rate of return is subtracted (currently hard–coded 5) from annualized average return and then divided by annualized standard deviation of returns.

**K–Ratio** – Detects inconsistency in returns. Should be 1.0 or more. The higher K ratio is the more consistent return you may expect from the system. Linear regression slope of equity line multiplied by square root of sum of squared deviations of bar number divided by standard error of equity line multiplied by square root of number of bars. More information: Stocks & Commodities V14:3 (115–118): Measuring System Performance by Lars N. Kestner

## OLD BACKTESTER REPORT



The screenshot shows a window titled "AmiBroker System Test Report - WinView". Inside, there is a table titled "Overall performance summary". The table contains various performance metrics for a trading system, including net profit, return on account, buy-and-hold performance, drawdowns, number of trades, and average trade results.

Overall performance summary			
Total net profit:	4365.37	Total commissions paid:	0.00
Return on account:	43.65 %	Open position gain/loss	468.44
Buy-and-hold profit:	-4943.62	Bars in test:	149
Buy-and-hold % return:	-49.42%	System to Buy-and-Hold index:	188.34%
Max. trade drawdown:	-2868.71	System drawdown:	-378.90
Total number of trades:	2	Percent profitable:	50.0%
Number winning trades:	1	Number losing trades:	1
Profit of winners:	5558.52	Loss of losers:	-1661.59
Total # of bars in winners:	82	Total # of bars in losers:	11
Commissions paid in winners:	0.00	Commissions paid in losers:	0.00
Largest winning trade:	5558.52	Largest losing trade:	-1661.59
# of bars in largest winner:	82	# bars in largest loser:	11
Commission paid in largest winner:	0.00	Commission paid in largest loser:	0.00
Average winning trade:	5558.52	Average losing trade:	-1661.59
Avg. # of bars in winners:	82.0	Avg. # bars in losers:	11.0
Avg. commission paid in winner:	0.00	Avg. commission paid in loser:	0.00
Max consec. winners:	0	Max consec. losers:	0
Ratio avg win/avg loss:	3.35	Avg. trade (win & loss):	1948.47
Profit factor:	3.35		

This window (accessible from **Report** button in [Automatic analysis window](#)) provides very useful information about the performance of a trading system under the test. The information included here can be customized using [system test settings dialog](#).

Explanation of values:

**Total net profit:** This is total profit/loss realized by the test. Includes the closed-out value of the open position (if there is any).

**Return on account:** This is total profit/loss as a percentage of initial investment.

**Total commissions paid:** The amount of commissions paid during trades.

**Open position gain/loss:** The closed-out value of open position that existed at the end of the test.

**Buy-and-hold profit:** The total profit/loss realized by buy-and-hold strategy (including commission).

**Buy-and-hold % return:** The total buy-and-hold strategy return as a percentage of initial investment.

**Bars in test:** The number of bars tested (Overall summary shows sum of number of bars in all symbols).

**Days in test:** The number of days between first bar date and last bar date (overall summary shows arithmetic average of number of days accross the population of symbols under test)

**System to buy-and-hold index:** An index showing how much better/worse is the system compared to buy-and-hold strategy. A value of 0% means that system gives the same profit as buy-and-hold strategy. A value of 200% means that system gives 200% more profit than buy-and-hold strategy. A value of -50% means that system gives a half of the gains of buy-and-hold strategy.

**Annual system % return:** Calculated compound annual percentage return of the system (\*see the note)

**Annual B&H % return:** Calculated compound annual percentage return of the buy and hold strategy (\*see the note)

**System drawdown:** The largest equity dip experienced by the system (relative to the initial investment).

**B&H drawdown:** The largest equity dip experienced by the buy and hold strategy (relative to the initial investment).

**Max. system drawdown:** The largest point distance between equity peak value and the following trough value experienced by the system

**Max. system % drawdown:** The largest percentage distance between equity peak value and the following trough value experienced by the system

**Max. B&H drawdown:** The largest point distance between equity peak value and the following trough value experienced by the buy and hold strategy

**Max. B&H % drawdown:** The largest percentage distance between equity peak value and the following trough value experienced by the buy and hold strategy

**Trade drawdown:** The largest equity dip experienced by any single trade (relative to the trade's entry price).

**Max. trade drawdown:** The largest point distance between equity peak value and the following trough value experienced by any single trade

**Max. trade % drawdown:** The largest percentage distance between equity peak value and the following trough value experienced by any single trade

**Total number of trades:** The number of trades (winners + losers)

**Percent profitable:** The number of winning trades compared to total number of trades shown as a percentage

**Profit of winners/Loss of losers:** Total amount of money gained in winners/lost in losers.

**Total # of bars in winners/losers:** The number of bars spent during winning/losing trades

**Largest winning/losing trade:** The amount of biggest winner/loser

**# of bars in largest winner/loser:** The number of bars in the biggest winning/losing trade

**Average winning/losing trade:** The average of winning/losing trades (sum of winners/losers divided by a number of winning/losing trades)

**Average # of bars in winners/losers:** The average of number of bars in winning/losing trades (total number

of bars in winners/losers divided by a number of winning/losing trades)

**Max consec. winners/losers:** The largest number of consecutive winning/losing trades.

**Bars out of the market:** The number of bars for which the system was completely out of the market (was neither long nor short). If you open and close the position during single day, even if you have no open position on market open and no position on close this day is NOT considered as out of the market.

**Interest earned:** The total interest earned between trades. Note that AmiBroker simulates O/N (overnight) deposits. This means that if you closed the position on Monday and opened the next one on Tuesday you earn interest for single O/N deposit.

**Exposure:** Shows how much you are exposed to the market. It is a ratio of bars in the market divided by total number of bars under test. (The number of bars in the market is given by total number of bars minus bars out of the market)

**Risk adjusted ann. return:** Shows annual return of the system (\*see note) adjusted (divided) by market exposure. If your system gained 10% over one year with the exposure of 50% the adjusted return would be 20% (10%/0.5)

**Ratio avg win/avg loss:** The absolute value of the ratio of average winning trade to average losing trade

**Profit factor:** The absolute value of the ratio of the profit of winners to loss of losers

**Avg. trade (win & loss):** The average trade profit calculated as sum of winners and losers divided by the number of trades.

\*Note: Calculation method used for annual percentage returns:

Most of the software (including two the most popular so-called professional packages) use very simple annualization method based on the following formula:

$\text{simple\_annualized\_percentage\_return} = \text{percentage\_return} * (365 / \text{days\_in\_test});$

unfortunately this method is **wrong** and very misleading since it would tell you that annual return is 22% when your system earned 44% during two years. This value is too optimistic. In fact annual return in this case is only 20%: if your initial investment was 10000 you earn 20% during the first year so you then get 12000 and 20% the second year that gives you 14400 = (12000 \* 120 %). So after two years you earned 44% but annually it is only 20%.

AmiBroker is one of the few programs that calculates annual returns correctly and will give you correct value of 20% as shown in the example above. The formula that AmiBroker uses for annual return calculation is as follows:

$\text{correctly\_annualized\_perc\_return} = 100\% * ((\text{final\_value}/\text{initial\_value}) ^ (365 / \text{days\_in\_test}) - 1)$

where  $x^y$  means rising x to the power of y.

#### **Known differences between statistics produced by 'old' and 'new' (portfolio) backtester**

	Old backtester	New (portfolio) backtester
System and trade drawdown calculations based	Open/Close/H-L range (worst case) selectable in settings	Close price only (regardless of settings) – subject to change



on		
Max. % trade drawdown	Calculated based on total equity	Calculated based on ACTUAL trade value at entry point.
Stats available	for all trades only	separately for long, short and all trades
PositionSizing	Based on individual symbol equity	Based on portfolio equity.  PositionSize = -25;  will enter 25% of current portfolio equity
Trade statistics	Include only closed trades, open trade is reported separately	Include all trades (closed and those still open at the end of analysis period). Any open trades are closed out at 'close' price always.
Exposure	calculated regardless of position size (no matter on what is position size if trade is taken for particular bar it assumes 100% exposure at that bar)	calculations include now (in 4.43.0) the total amount of open positions compared to total portfolio equity. Exposure is calculated on bar by bar basis so if only 50% funds are in open trade, then exposure for this bar is 0.5. Then individual bar exposures are summed up and divided by number of bars to produce exposure figure. This way true market exposure is calculated.
Multiple security testing	N independent accounts (multiple single equity)	Portfolio equity common to all symbols under test

## Commission window

**Commission Table**

<b>Default</b>	
Per share	0,005
Per trade	0
% of trade value	
Min amount	1
Min % of value	
Max amount	
Max % of value	0,5
<b>Tier 1</b>	
Tier Unit	Trans. Value
Tier Minimum	

**Per share**  
Per-share commission amount in base currency. Use zero when commission schedule does not have per-share

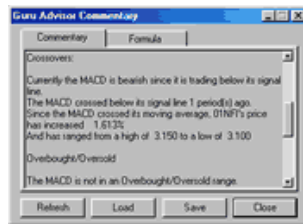
Commission table is available in the [Account manager](#) and in [Automatic Analysis → Settings](#) window, "General" tab, "Commission and rates: Define..."

In this window you can enter commission taken by buy/sell transactions.

There are 5 tiers of commission schedule table plus "default" tier that is used when others are not defined or transaction does not match any tier defined. Tiers can be defined based on transaction value or number of shares/contracts traded. Each tier has user definable minimum and maximum. If min/max is not defined or set to zero – the tier is not active.

Each tier allows to define commission on per–share, per–trade, % of trade volume basis and allows to define minimum and maximum commission values based on dollar or percent values.

## Commentary window



Commentary window enables you to view textual descriptions of actual technical situation on given market.

Commentaries are generated using formulas written in AmiBroker's own formula language. You can find the description of this language in [AmiBroker Formula Language Reference Guide](#).

Moreover Commentary feature gives you also graphical representation of buy & sell signals by placing the marks (arrows) on the price chart.

Newbies should read "[Tutorial: How to write your own commentary](#)" for step–by–step instructions and [working with AFL editor](#).

"Refresh" button causes AmiBroker to reinterpret the commentary using currently selected symbol/date.

"Load" and "Save" buttons allow to load/save commentary formulas.

"Close" button closes the commentary window.

Now the Guru chart commentary window is automatically updated and synchronized with the date selected on the chart using "Pick" selector tool. This way you can easily read any indicator value on any selected date right off the chart commentary window.

## Plugins window

Plugins window lists all loaded plug in DLLs. It is useful for inspecting which plugins are active.

In addition to just showing the list of plugins you can unload all DLLs by pressing "Unload" button and load them back by pressing "Load" button. Please note that a DLL **must** be placed in the "Plugins" subfolder of AmiBroker main directory to be seen.

At start AmiBroker scans the "Plugins" folder and loads the DLLs that follows the specifications of AmiBroker plugin. If a DLL is loaded it is "locked" for writing so it can not be overwritten or modified.

During the development process it is necessary to overwrite/modify the DLL code – because when you apply the changes to the source code these changes must be recompiled and stored into DLL file. To allow the developer to overwrite the DLL used by AmiBroker the "Unload" function is available in this window. Unloading releases the DLL so it can be overwritten without the need to restart AmiBroker. Then, after

modifying the DLL code, you can load the DLL back using "Load" function.

**IMPORTANT NOTE:** AmiBroker makes no representations on features and performance of non-certified third-party plug-ins. Specifically certain plug-ins can cause instabilities or even crashes. Entire use of non-certified third-party plugins is at your own risk.

## Indicator Maintenance Wizard

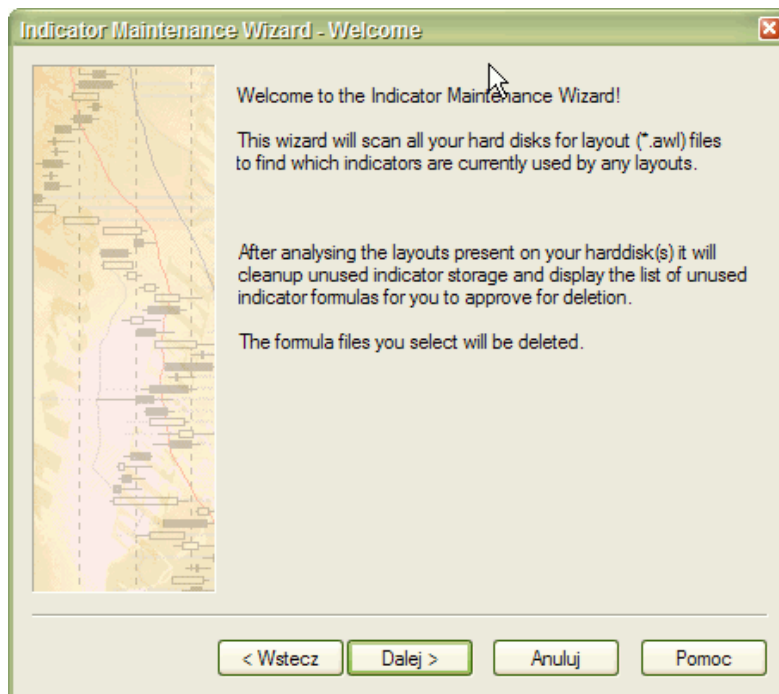
Indicator maintenance checks for any indicators that were deleted from any layouts on your hard disk and frees table from entries

allocated for indicators that were deleted. This procedure is rarely but still needed because if you delete indicator from one

layout there is no guarantee that there is no other layout file buried somewhere on your hard disk that still references given indicator.

So Indicator Maintenance scans all hard disks and all partitions looking for layout files and analysing them to built the

" actually used" table of indicators.



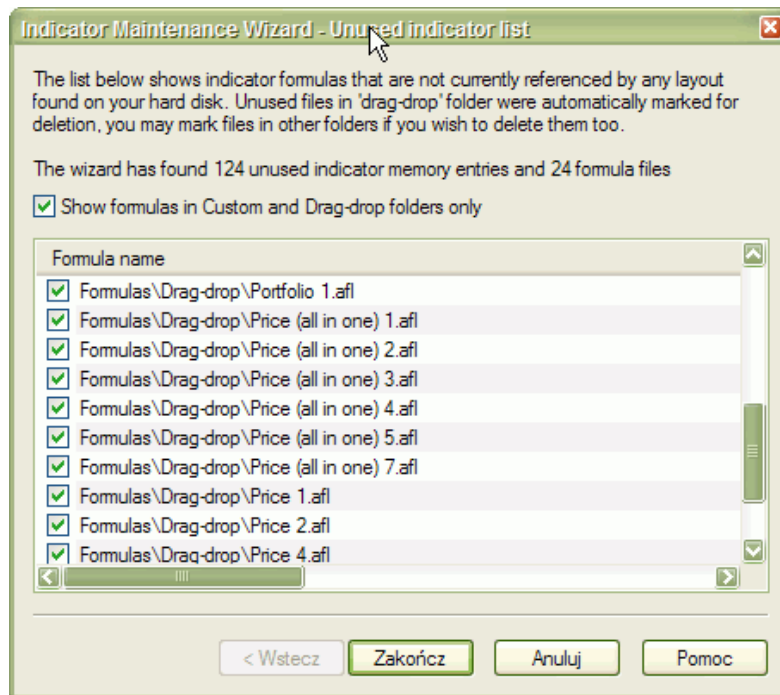
The ones which are not referenced by any layout can be deleted from internal table.

Depending on your choice you may leave default behaviour (cleaning up only internal table) or deleting actual formula files

that are not referenced. This is up to you. If you don't use particular formula for say AA

Scan/Backtest/Optimization you can

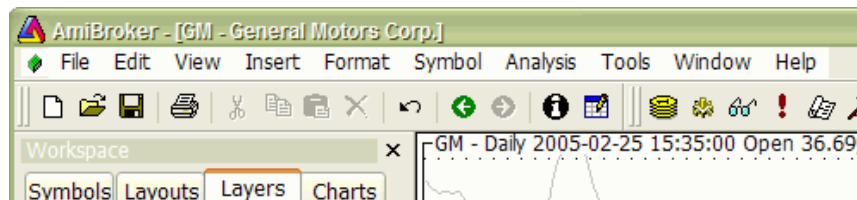
delete it. If you use it or need it for some archive purposes – leave it unchecked.



If you are not sure what options to choose, just press "Next" all the time and you will safely complete the procedure without changing any settings.

## Menus

This chapter describes AmiBroker menus.



There are following main pull down menus:

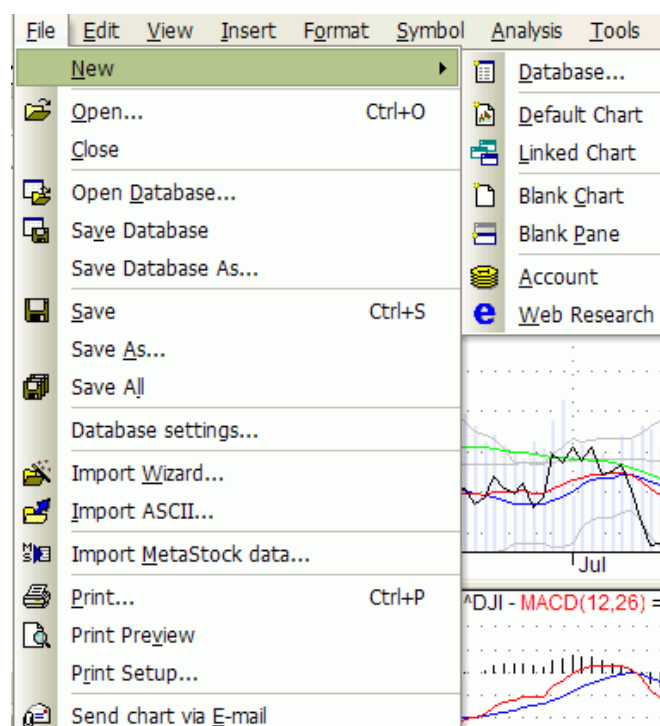
- [File](#)
- [Edit](#)
- [View](#)
- [Insert](#)
- [Format](#)
- [Symbol](#)
- [Analysis](#)
- [Tools](#)
- [Window](#)
- [Help](#)

And the following CONTEXT menus:

- [AFL Editor context menu](#) (available when you click with RIGHT mouse button in the AFL editor)

- [Automatic Analysis window context menu](#) (available when you click with RIGHT mouse button over Automatic Analysis RESULT LIST)
- [Alert Output context menu](#) (available when you click with RIGHT mouse button in the Alert Output window)
- [Chart pane context menu](#) (available when you click with RIGHT mouse button in the chart pane)
- [Layouts context menu](#) (available when you click with RIGHT mouse button in the Workspace → Layouts tree )
- [Formula context menu](#) (available when you click with RIGHT mouse button in the Workspace → Charts tree )
- [Layers context menu](#) (available when you click with RIGHT mouse button in the Workspace → Layers list )
- [RealTime Quote context menu](#) (available when you click with RIGHT mouse button in the Real time quote list)

## File menu



### New

- **Database**  
Creates a new AmiBroker database and launches [Database settings](#) window.
- **Default Chart**  
Creates new chart window using default template. It's possible to select the symbols and time frame independently in each of the windows opened.
- **Linked Chart**  
Creates linked chart window based on current template and active chart. Linked windows use the same symbol selection, so if you change the selected symbol for one of them, the other one will synchronize automatically. Linked windows can have DIFFERENT viewing time frame selected. Simply activate the window and select desired interval from [View](#) menu for one window, then switch to the other one and select different interval for it. This option allows you to select different time frame or indicators' set in each window and easily move through the database.

- **Blank Chart**

Creates new (blank) chart window. This is useful if you want to create completely new setup of charts that do not share the same chart IDs. It is important if you want to have indicators that have independent parameters from the other windows that you have created.

- **Blank Pane**

Creates new (blank) chart pane

- **Account**

Creates New [Account \(Account Manager\)](#)

- **Web Research**

Creates New [Web Research](#) window

**Open**

Opens document (account, database or HTML file – you can pick document type from "Files of type" combo in the File selector window)

**Close**

Closes current (active) document window (chart, account, web research)

**Open Database**

Allows you to open an existing AmiBroker database. Please select the database folder and press OK.

**Save Database**

Saves the currently used database

**Save Database As...**

Saves database into new location

**Save**

Saves current document (account, html file)

**Save As...**

Saves current document (account, html file) under new name

**Save All**

Saves all documents currently open

**Database Settings**

Opens [Database settings](#) dialog that allows you to change your database parameters or intraday settings.

**Import Wizard**

Launches ASCII [Import Wizard](#) window, that allows you to easily import ASCII (text) files into your database

**Import ASCII**

Allows you to import ASCII files with use of predefined import formats. To learn more how to use ASCII importer, please read [ASCII Importer reference](#) chapter.

**Import MetaStock data**

Launches [Metastock importer](#) window. *IMPORTANT NOTE: Metastock importer should be used ONLY if you want to import MS data to native, local AmiBroker database once. If you want AmiBroker to just read Metastock database **DIRECTLY** without need to import new data over and over please set up your database [WITH METASTOCK PLUGIN](#) as described in the Tutorial.*

**Print**

Allows you to print currently displayed charts.

**Print Preview**

Prints currently displayed charts with the preview (you can check the appearance of the document before it's printed).

**Print Setup**

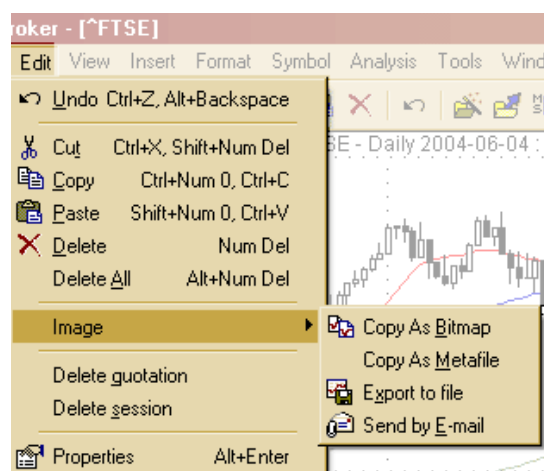
Opens printout setup dialog.

**Send Chart via E-mail**

AmiBroker creates .png image (with the currently displayed chart) and uses your default mailing program (e.g. Outlook Express) to send the file as an attachment.

**Exit**

Closes AmiBroker program.

**Edit menu****Undo**

Allows to undo the last operation performed on chart studies (trendlines etc.). This option will be unavailable if no study has been drawn or moved.

**Cut, Copy, Paste, Delete**

These options can be used to cut, copy, paste or delete studies from the chart. Cut, copy and delete will be greyed out if no object on the chart is selected. To paste the object, it's necessary to use 'copy' or 'cut' option first.

To learn more about drawing tools in AmiBroker, please read [Drawing tools reference](#) chapter.

**Delete All**

Deletes all the objects from the currently opened chart window.

**Image**

- **Copy As Bitmap** – copies the currently opened chart to the system clipboard as a .BMP image. You can paste the clipboard contents e.g. into 'Paint' application.

- **Copy As Metafile** – copies the currently opened chart to the system clipboard as a metafile
- **Export to file** – saves the currently displayed chart as .PNG file
- **Send by E-mail** – AmiBroker creates .png image (with the currently displayed chart) and uses your default mailing program (e.g. Outlook Express) to send the file as an attachment.

### Delete quotation

Deletes currently selected bar.

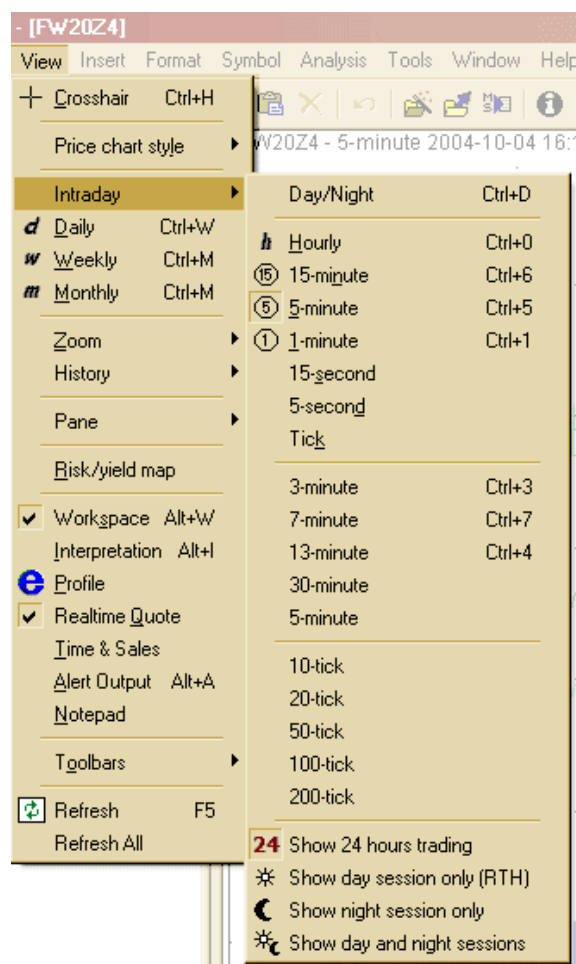
### Delete Session

Deletes currently selected bar from ALL the symbols in the database.

### Properties

Opens a study properties dialog. More information can be found in [Drawing tools reference](#) chapter.

### View menu



### Crosshair

Turns on/off crosshair.

### Price Chart Style

Changes the style of the default Price chart

- **Auto** – uses settings defined in Tools -> Preferences
- **Line** – line chart
- **CandleSticks** – candlestick chart
- **Bars** – traditional bar chart

### Intraday

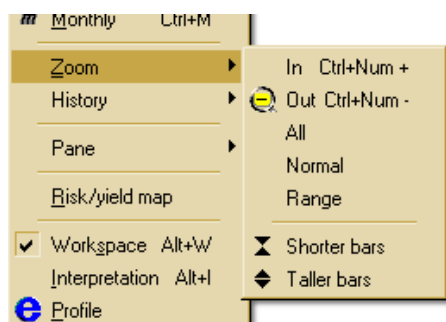
Allows you to choose one of intraday time intervals and decide whether to display day or night sessions. Day and night sessions' hours can be set in: [Database settings](#) window (*File -> Database Settings -> Intraday settings*) or separately for group in [Categories window](#) (*Symbol -> Categories*).

- **Day / Night** – shows two bars (day and night) per day
- **Show 24 hours trading** – no filtering is applied and all the data in the database is included in the chart.
- **Show day session only** – displays day sessions only.
- **Show night session only** – displays night sessions only.
- **Show day and night sessions** – displays day and night sessions.

### Daily, Weekly, Monthly

Allows to change the display time interval.





## Zoom

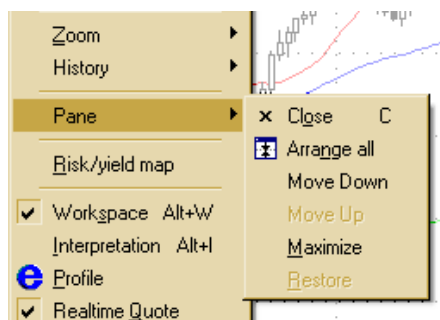
Controls the zoom of the chart

- **In** – reduces number of bars displayed
- **Out** – increases number of bars displayed
- **All** – displays all the available bars for the current symbol
- **Normal** – displays default number of bars (defined in Tools -> Preferences -> Charting)
- **Range** – displays the bars from the selected range
- **Shorter bars** – reduces the vertical size of the bars
- **Longer bars** – increases the vertical size of the bars

## History

Allows to move Back/Forward in 'browser-like' way.

- **Previous** move to previous symbol (keyboard shortcut: Ctrl+Alt+LEFT)
- **Next** move to next symbol (keyboard shortcut: Ctrl+Alt+RIGHT)



## Pane

- **Close** – closes currently selected chart pane
- **Arrange All** – arranges all the displayed charts
- **Move Down** – moves currently selected chart pane one position down
- **Move Up** – moves currently selected chart pane one position up
- **Maximize** – maximizes currently selected chart pane
- **Restore** – restores the charts layout after using Maximize

## Risk/Yeld map

Displays Risk/Yeld map of all the symbols in the database. Risk/yield map calculates average weekly return (the yield) and standard deviation of the weekly returns (the risk) over at least 12 weeks. It requires at least 60 bars worth of data for every stock. To zoom in – mark the area with the mouse. To zoom-out simply click on the map.

## Workspace

Displays/hides Workspace window. The window contains:

- **Symbols** tab – symbols tree with categories (See: [Understanding categories](#)).

- **Layouts** tab – list of available global and local layouts (See: [Working with chart sheets and window layouts](#)).
- **Layers** tab – list of chart layers (See: [Working with layers](#)).

**Interpretation**

Displays/hides the Interpretation window.

**Profile**

Displays/hides the Symbol Profile View window. More: [How to set up the profile view](#).

**Realtime Quote**

Displays/hides the Realtime Quote window. The RT quote window provides real-time streaming quotes and some basic fundamental data. To learn more read: [How to use AmiBroker in Real Time mode](#) chapter.

**Alert Output**

Shows/hides Alert Output window. The window displays texts generated by formula based alert. The detailed information on how to use alerts is available in: [Using formula-based alerts](#) part of the Users' Guide.

**Notepad**

Displays/hides Notepad window, that allows to store free-text notes about particular security. Just type any text and it will be automatically saved / read back as you browse through symbols. Notes are global and are saved in "Notes" subfolder as ordinary text files.

**Toolbars**

Allows you to display/hide the toolbars.

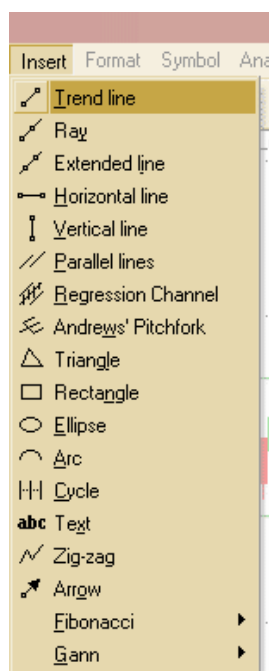
**Refresh**

Refreshes the chart window.

**Refresh All**

Refreshes the chart window and re-reads the contents of all the categories in symbols tree in Workspace window.

**Insert menu**



### **Trend line**

Draws a trend line.

To draw a trend line the chart – start drawing by pointing the mouse and pressing left mouse button where you want to start the drawing. Then move the mouse and study tracking line will appear. Release left mouse button when you want to finish drawing. Alternatively you can click once in the place where you want the trendline to begin, move the mouse and click once again to finish drawing. You can also cancel study drawing by pressing ESC (escape) key.

### **Ray**

Draws a ray. Ray is a right-extended trend line.

### **Extended line**

Draws an extended line. Extended line is a trend line that is extended automatically from both left- and right-sides.

### **Horizontal line**

Draws a horizontal line. Horizontal line is self expanding so it is only necessary to click on the chosen price-level.

### **Vertical line**

Draws a vertical line. Vertical line is self expanding so it is only necessary to click on the chosen bar.

**Parallel lines**

Draws parallel trendlines.

This tool allows to draw a series of parallel trend line segments. First you draw a trend line as usual, then a second line parallel to the first is automatically created and you can move them around with the mouse. Once you click on the chart it is placed in given position. Then another parallel line appears that can be placed somewhere else. And again, and again. To stop this please either press ESC key or choose "Select" tool.

**Regression channel**

Draws Raff, standard deviation, standard error channels. To read the detailed information regarding this tool please read [Drawing tools reference](#) chapter.

**Andrews' pitchfork**

Draws an Andrews' pitchfork. Read [Drawing tools reference](#) chapter for more detailed information.

**Triangle**

Draws a Triangle. Left-click in the first point, move to the second point then click once, then move to the third point and click once again.

**Rectangle**

Draws a rectangle. Left-click in the first point, move to the position where you want to place the opposite corner and click once again.

**Ellipse**

Draws an Ellipse. Ellipse is connected to the date/price coordinates (as trend lines) rather than to the screen pixels so it can change the visual shape when displayed at various zoom factors or screen sizes. To see the properties of ellipse you should double-click on the clock-like 3, 6, 9 or 12 hour positions.

**Arc**

Draws an Arc. Arc, the same as Ellipse is connected to the date/price coordinates (as trend lines) rather than to the screen pixels so it can change the visual shape when displayed at various zoom factors or screen sizes. To see the properties of ellipse you should double-click on the clock-like 3, 6, 9 or 12 hour positions.

**Cycle**

Draws time cycles. To use time cycles tool, click on the cycles drawing tool button in the toolbar then click at the starting point of the cycle and drag to the end of the cycle. These two control points control the interval between the cycle lines. When you release the mouse button you will get a series of parallel lines with equal interval in between them.

**Text**

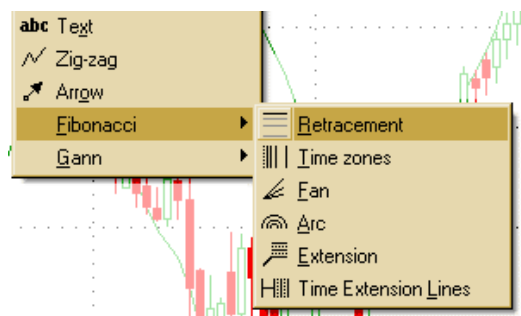
Allows to place a custom text on the chart. Left-click on the chart to start typing. To finish – click once again on the chart, outside the text box. You can also cancel typing by pressing ESC (escape) key.

**Zig-zag**

Draws a series of connected trend lines. To finish the series double-click or press ESC (escape) key.

**Arrow**

Draws a line that ends with an arrow. Drawing technique is exactly the same as drawing a trend line.

**Fibonacci**

Group of Fibonacci drawing tools. Read [Drawing tools reference](#) chapter for more detailed information.

- Fibonacci Retracement study
- Fibonacci Time zones study
- Fibonacci Fan
- Fibonacci Arc
- Fibonacci Extensions
- Fibonacci Time Extension lines

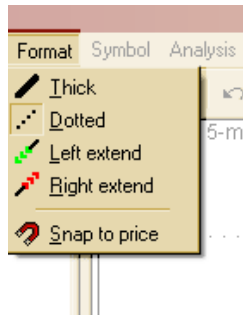
**Gann**

Group of Gann drawing tools.

- Gann Fan
- Gann Square

Read [Drawing tools reference](#) chapter for more detailed information.

## Format menu



These options allow you to apply color or style to the objects. Note that you can also select color and style of the object before drawing new object: simply deselect previous object (if any), change style selections and draw new object.

### Thick

Changes drawn object formatting to thick style.

### Dotted

Changes study formatting to dotted style.

### Left Extend

Extends the trendline to the left.

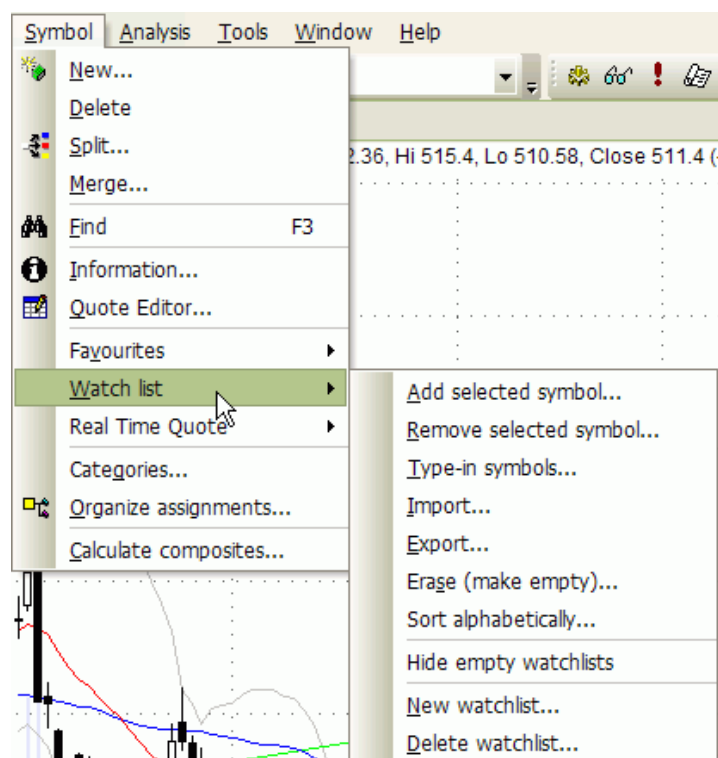
### Right Extend

Extends the trendline to the right.

### Snap to price

Turns on the magnet that snaps the drawn studies to the prices. Snap to price % threshold can be set in [Preferences](#) window. Snap to price % threshold defines how far price 'magnet' works, it will snap to price when the mouse is nearer than % threshold from H/L/C price

## Symbol menu



### New

Allows you to add new symbols into the database. After selecting this function you will be prompted for new ticker symbol. Please try not to exceed 26 chars. For proper import functioning you should enter the symbol with CAPITALS.

### Delete

Removes currently selected symbol from the database. After choosing this function you will be asked for confirmation of symbol removing. Note well that this operation can not be undone.

### Split

Allows you to perform stock split. AmiBroker provides easy way of handling stock splits. Program will try to guess split date and ratio by analyzing quotations. If there is just a single quotation after split this should work, if not you will be asked for split date and ratio. You can specify a split using following expression:  $x \rightarrow y$  which means that  $x$  shares before split become  $y$  after it. For example  $2 \rightarrow 3$  means that 2 shares become 3 after the split. It is also possible to perform reverse-split, for example  $2 \rightarrow 1$ , which means that 2 shares are joined together into 1 share.

### Merge

This function allows you to merge two tickers, when the ticker for the symbol is changed and in your database – one symbol holds historical quotes and the second one holds newest quotes (after name change). You should just select the new ticker (after name change) and use *Symbol* → *Merge*. Then from the combo you should choose original ticker ("merge with") and optionally check the following fields:

- overwrite duplicate quotes – checking this option will overwrite the quotes already existing in "new" ticker with those present in "old" ticker (this should really not be the case, but may happen).
- delete "merge with" afterwards – checking this option will delete the "old" ticker after merging
- assign alias name – checking this option will copy the "old" ticker to the alias field of the "new" ticker

**Find**

Opens [Symbol finder](#) window that allows you to quickly search the database for a symbol by typing the first letters of its full name or ticker.

**Information**

Opens the [Information window](#) for the symbol, which allows you to change the symbol properties.

**Finances**

[Finances window](#) allows you to enter some fundamental data for the symbol (sales income, earnings before taxes (EBT), earnings after taxes (EAT) ). AmiBroker will compute P/E (Price to Earnings ratio) and EPS (Earnings Per Share) indicators out of the data given.

**Quote Editor**

Opens [Quote Editor](#) window that allows you to edit, delete and add quotations into your database.

**Watchlist**

These options allow you to manage your watchlists. [Working with watch lists](#) chapter explains in more detail the way you can use the below options.

- **Add Selected Symbol** – adds the currently selected symbol to the specified watchlist(s).
- **Remove Selected Symbol** – removes the currently selected symbol from the specified watchlist(s).
- **Type-in Symbols** – allows you to type-in the symbols to the watchlist(s).
- **Import** – allows to import the watchlist from the .TLS file
- **Export** – exports the symbols belonging to the watchlist to the .TLS (symbol list) file
- **Erase (make empty)** – removes all the symbols from the specified watchlist.
- **Sort alphabetically** – sorts tickers alphabetically in the specified watchlist
- **Hide empty watchlists** – hides watch lists with no symbols in the symbol tree
- **New watchlist** – creates new watch list
- **Delete watchlist** – deletes selected watch list (it does not delete symbols from the database)

**Categories**

[Categories window](#) allows you to define names of markets, groups, sectors and industries. For each market you can also define base indexes for calculating relative strength, composite data, beta or web profile URL.

**Organize assignments**

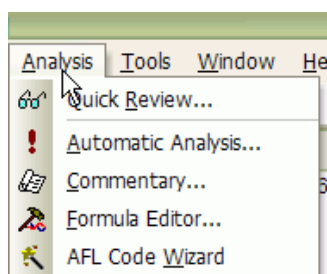
[Assignment organizer window](#) allows you to easily change the category assignments for the symbols or to delete multiple symbols from the database.

**Calculate Composites**

Opens [Composite calculation window](#) that allows for automatic calculation of number and volume of advancing/declining/unchanged issues or volume numbers for indices.

**Analysis menu**





### Quick Review

Opens [Quick review](#) window that provides overall market information like: daily symbol quotes, daily/weekly/monthly/quarterly/yearly returns comparison table or Price/Earnings and Price/Book value comparison.

### Automatic Analysis

Opens [Automatic Analysis](#) window that enables you to check your quotations against defined buy/sell rules or explore your database. AmiBroker can produce report telling you if buy/sell signals occurred on given symbol in the specified period of time, simulate trading, giving you an idea about performance of your system or optimize the trading system you use to improve it's performance.

### Commentary

Displays [Commentary](#) window which allows you to view textual descriptions of actual technical situation on given market.

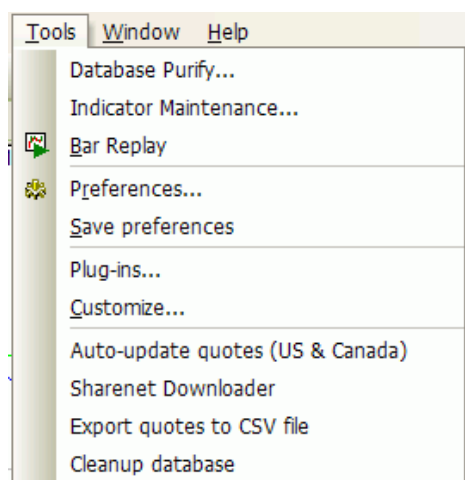
### Formula Editor

Opens the [Formula Editor](#) window that enables you to write your own formulas.

### AFL Code Wizard

Opens the AFL Code Wizard – the add-on program that creates trading system AFL code from plain English sentences. See [introduction video to AFL Code Wizard](#).

## Tools menu



### Database Purify

Database purify tool allows to detect missing/extra quotes, possible splits or invalid OHLC relationship.

**Indicator Maintenance**

Opens [Indicator Maintenance wizard](#), that helps to clean up unused indicator space

**Bar Replay**

Opens [Bar Replay](#) tool, which allows to replay historical data.

**Preferences**

Opens [Preferences](#) window which allows you to configure the program.

**Save Preferences**

Saves all the preferences changes (the information is store in *broker.prefs* file).

**Plugins**

Opens [Plugins](#) window. It contains the lists of all loaded plug-in DLLs and can be used for inspecting which plugins are active. It's also possible to unload the plugins.

**Customize**

[Customize tools](#) dialog allows you to define custom tools that can be invoked from Tools menu.

**Auto-update quotes**

Auto-update quotes option updates historical quotes from the last date present in AmiBroker upto today with use of AmiQuote Downloader. The detailed description on how to use AmiQuote do obtain free quotations can be found in [Automatic update of EOD quotes](#) tutorial chapter.

**Sharenet Downloader**

Launches the script which downloads the quotations from Sharenet (South Africa only).

**Export to CSV file**

Runs a script that exports the database to the CSV file. Note that you can use [Automatic Analysis](#) window to export the quotes way faster than with use of this script.

**Cleanup database**

Launches the script that allows you to find non-traded stocks in the database. Script automatically scans the database and checks the latest quotation date. If it is old enough, the script will display warning message and lets you decide whether the stock should be deleted or not. Additionally script can generate a list of "old" stocks and save it to the text file. The detailed information is available in: [05-2000 issue of the newsletter](#).

**Window menu**

**IMPORTANT NOTE** to old version users: **Window -> New** and **Window -> New Linked** options have been moved to **File->New->Default Chart** and **File->New->Linked Chart** menus.

**Cascade**

Cascades opened chart windows.

**Tile Horizontally**

Tiles the opened chart windows horizontally.

**Tile Vertically**

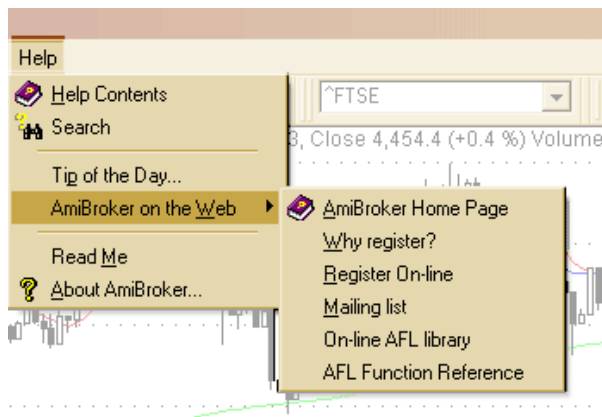
Tiles the opened chart windows vertically.

**Arrange Icons**

Allows you to arrange the minimized windows. Arrange icons works only if:

- You created more than two windows (via Window->New or Window->New Linked)
- You have minimized them
- You moved the minimized boxes

Arrange icons option will align the windows nicely at the bottom of the AmiBroker window.

**Help menu****Help Contents**

Displays the contents of the AmiBroker Users' Guide.

**Search**

Allows you to search the Users' Guide.

**Tip of the day**

Shows *Tip of the day* dialog where many useful usage tips are displayed.

**AmiBroker on the web**

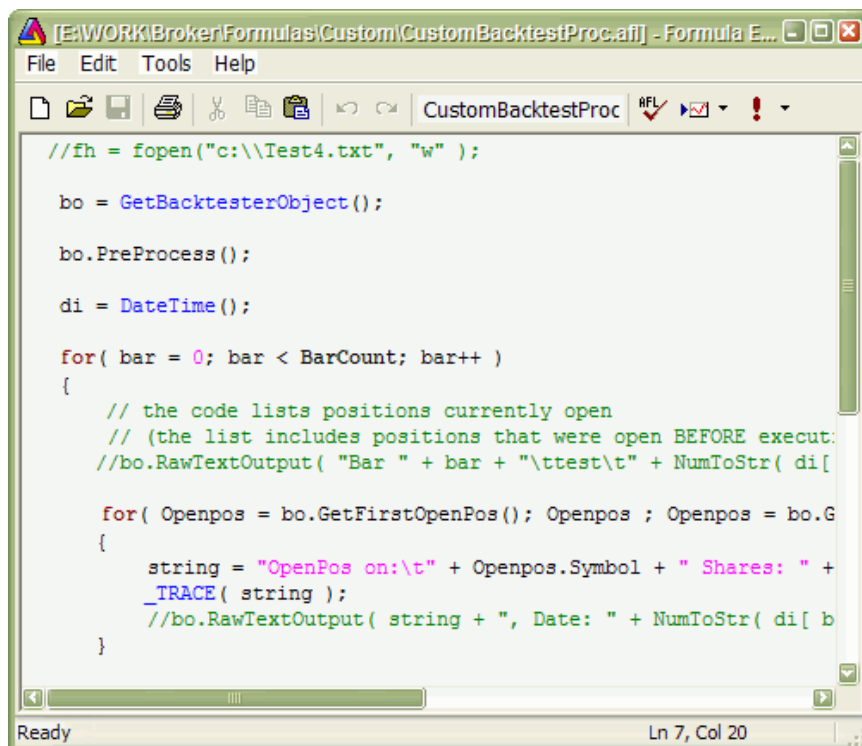
- link to [AmiBroker Home page](#)
- list of [benefits for registered users](#)
- secure [On-line order form](#)
- [AmiBroker Mailing List](#)
- [On-line formula library](#)
- [On-line AFL function reference](#)

**Readme**

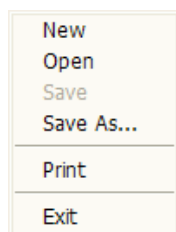
Displays the contents of Readme file. Please note that all the recent changes in beta releases are reported in Readme.

**About AmiBroker**

Shows the 'About' window, which contains the information about program version and user details.

**AFL Editor menu**

AFL editor features separate menu consisting of the following choices:

**1. File**

where

- New – clears the formula editor window
- Open – opens the formula file
- Save – saves the formula under current name
- Save As.. – saves the formula under new name
- Print – prints the formula
- Exit – closes the editor

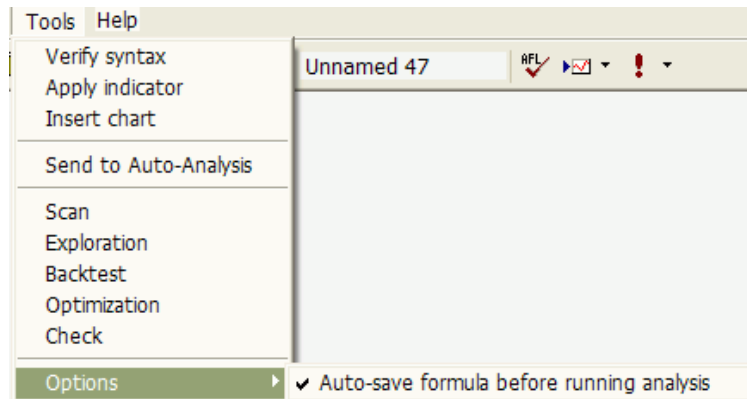
## 2. Edit

Undo	Ctrl+Z
Redo	Ctrl+Y
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Select All	Ctrl+A
Find...	Ctrl+F
Copy Error Message	

where

- Undo – un–does recent action (multiple–level)
- Redo – re–does recent action (multiple–level)
- Cut – cuts the selection and copies to the clipboard
- Copy – copies the selection to the clipboard
- Paste – pastes current clipboard content in the current cursor position
- Select All – selects entire text in the editor
- Find... – provides access to text search tool
- Copy Error Message – copies current error message displayed in the bottom of the editor window to the clipboard (option is active only when there are any errors displayed after syntax check)

## 3. Tools



where

- Verify syntax – checks current formula for errors
- Apply indicator – saves the formula and applies current formula as a chart/indicator ONCE
- Insert chart – saves the formula and applies current formula as a chart MANY TIMES (inserts multiple times)
- Send to Auto–Analysis – saves the formula and selects it as current formula in Automatic Analysis window
- Scan – saves the formula and performs Scan in Automatic Analysis window
- Exploration – saves the formula and performs Exploration in Automatic Analysis window
- Backtest – saves the formula and performs Backtest in Automatic Analysis window
- Optimization – saves the formula and performs Optimization in Automatic Analysis window

- Check – saves the formula and performs Check (if given formula references future) in Automatic Analysis window
- Options: Auto-save formula before running analysis – when checked, any click on Scan/Explore/Backtest/Optimize button in Automatic Analysis window triggers automatic save of current formula.

#### 4. Help

Function reference	F1
Parameter info	
AFL Language Reference	
Function index by Name	
Function index by Category	
Help on Editor	

where

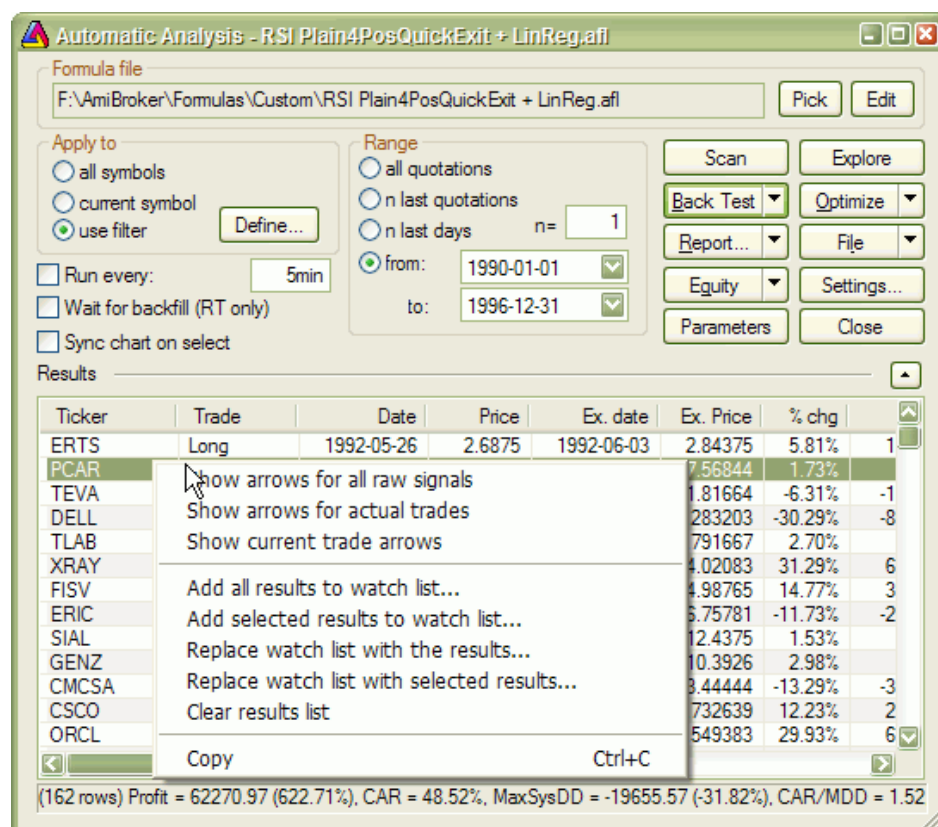
- Function reference – displays reference page for currently highlighted AFL function, more on this feature [here](#).
- Parameter info – displays parameter tooltip for currently highlighted AFL function, more on this feature [here](#).
- AFL Language reference – displays [language reference](#) page.
- Function index by Name – displays [alphabetical list of AFL functions](#).
- Function index by Category – displays [categorized list of AFL functions](#).
- Help on Editor – displays [this help page](#).

as well as context menu (available via RIGHT click over the formula):

Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Delete	Del
Undo	Ctrl+Z
Redo	Ctrl+Y
Select All	Ctrl+A
Find	Ctrl+F
Print	
Parameter info	
Function reference	F1

which essentially duplicates choices available from regular menu.

### Automatic Analysis result list context menu



This menu shows up when you click with RIGHT mouse button over Automatic Analysis result list.

Available choices:

- Show arrows for all raw signals – show buy/sell/short/cover arrows for all raw (unfiltered) signals. If your formula is for example  

$$\text{buy} = C > \text{MA}(C, 10);$$
you will get a buy (solid green) arrow for all bars where close was above 10–bar moving average
  - Show arrows for actual trades – show arrows only on trade entry/exit bars. This shows arrows for ALL TRADES. If your formula is for example  

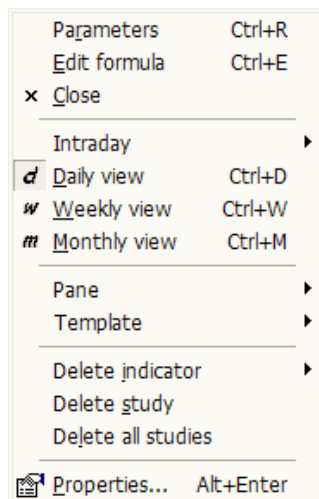
$$\text{buy} = C > \text{MA}(C, 10);$$
you will get a buy (solid green) arrow only for the very first bar when close crossed above moving average and trade was initiated, and you won't get any subsequent buy arrows until a matching sell (trade exit) occurs.
- Note that trade arrows represent all possible trades taken. Given trade may not be taken by backtester if there are insufficient funds to enter it.
- Show current trade arrows – show entry/exit arrows for selected trade only. This displays the arrows for currently selected trade (from the result list). It represents trade actually taken.
  - Add all results to watch list – adds all symbols from the result list to the watch list of your choice. More on this [here](#)

- Add selected results to watch list – adds symbols from selected rows to the watch list of your choice. More on this [here](#)
- Replace watch list with all results – empties the watch list and then adds all symbols from the result list to the watch list of your choice. More on this [here](#)
- Replace watch list with selected results – empties the watch list and then adds symbols from selected rows to the watch list of your choice. More on this [here](#)
- Clear result list – removes all rows from the result list
- Copy – copies result list to the Windows clipboard, so you can paste it to some other application, like Excel for example

#### IMPORTANT NOTES:

1. Buy arrow is solid green, Sell arrow is solid red, Short arrow is hollow red, Cover arrow is hollow green
2. Arrows are shown only on the charts that have "Show arrows" property turned ON.

#### Chart context menu



This context menu shows up when you click with RIGHT mouse button over chart pane.

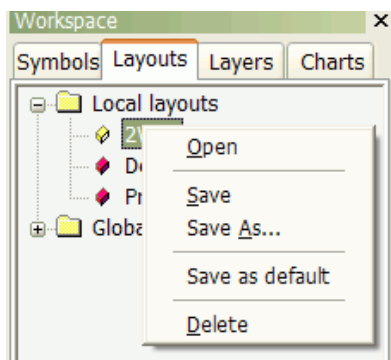
Available options:

- **Parameters...** – brings up [Parameters](#) dialog allowing you to modify parameters of indicators, as well as colors, styles, scaling and axes settings
- **Edit Formula...** – brings up [Formula Editor](#) allowing you to view/modify the AFL code of indicator
- **Close** – closes chart pane
- **Intraday ...** – allows you to switch viewing time frame to one of available intraday intervals
- **Daily view** – switches viewing interval to daily
- **Weekly view** – switches viewing interval to weekly
- **Monthly view** – switches viewing interval to monthly
- **Pane**
  - ♦ **Close** – closes chart pane
  - ♦ **Arrange all** – arranges panes to equal height
  - ♦ **Move up** – moves selected chart pane up (switches pane vertical order)



- ♦ **Move down** – moves selected chart pane down (switches pane vertical order)
- ♦ **Maximize** – maximizes selected pane so it fills entire screen
- ♦ **Restore** – restores selected pane to previous size
- **Template**
  - ♦ **Load...** – loads single window chart template from the selected file (more on templates and layouts [here](#))
  - ♦ **Save...** – saves single window chart template to the selected file
  - ♦ **Load default** – loads default single window template
  - ♦ **Save as default** – saves current single window setup as default template
- **Delete indicator** – deletes one of drag-and-drop indicator sections found in the code
- **Delete study** – deletes selected manually drawn study (like trend line, Fibonacci, Gann...) – more on this [here](#)
- **Delete All studies** – deletes all manually drawn studies (like trend line, Fibonacci, Gann...)
- **Properties** – displays properties (coordinates, colors, etc) of manually drawn study (like trend line, Fibonacci, Gann...) more on this [here](#) and [here](#).

## Layouts context menu



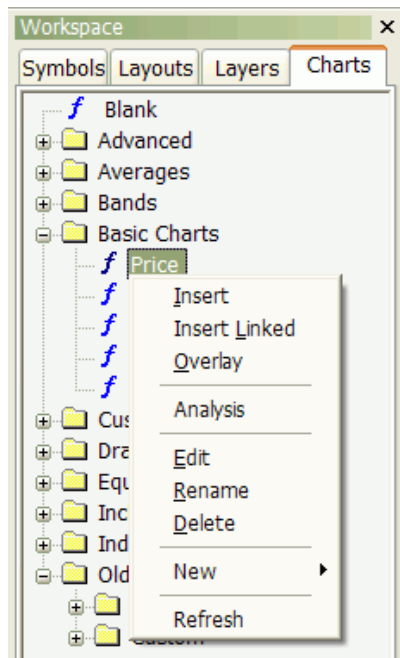
Layouts context menu shows up when you click with RIGHT mouse button over layout in the **Workspace** window, **Layouts** tab.

Available choices:

- Open – loads selected layout
- Save – saves current window layout under current name
- Save As... – save current window layout under new name
- Save as default – save current window layout as default (startup) layout for given database
- Delete – delete selected layout

To learn more about Layouts please check [Tutorial: Chart sheets and layouts](#)

## Formula (chart) context menu



Formula (chart) context menu shows up when you click with RIGHT mouse button over formula listed in the **Charts** tab of **Workspace** pane (see picture on the left)

Available choices:

- **Insert** – inserts selected indicator into new chart pane.

**Insert** command internally creates a copy of the original formula file and places such copy into hidden drag-drop folder so original formula will not be affected by subsequent editing or overlaying other indicators onto it.

Double clicking on formula name is equivalent with choosing Insert command from the menu.

- **Insert Linked** – inserts selected indicator into new chart pane directly (i.e. linked to original).

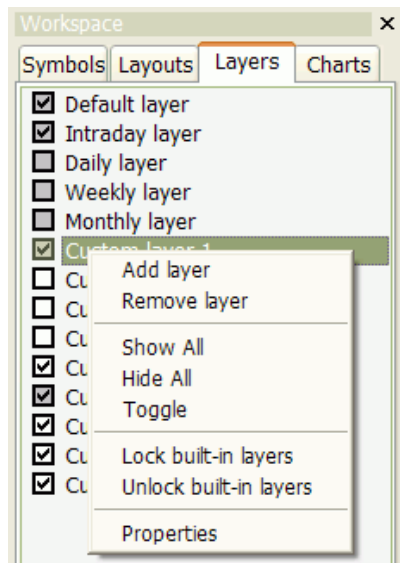
**Insert Linked** command does not create any copy of the formula. Instead it creates new chart pane that directly links to original formula. This way subsequent editing and/or overlaying other indicators will modify the original

- **Overlay** – overlay selected indicator onto selected chart pane

Overlay command internally appends additional code to the formula used by the chart pane. If given chart pane was created usign Insert Linked, it will modify original (linked) formula.

- **Analysis** – show up Automatic Analysis window and pick selected formula
- **Edit** – open Formula Editor window to edit selected formula
- **Rename** – rename currently selected formula file
- **Delete** – delete currently selected formula file
- **New**
  - ♦ **Formula** – creates new formula file in currently selected folder
  - ♦ **Folder** – creates new subfolder under currently selected folder
- **Refresh** – re-reads Formula directory and re-display formula tree

## Layers context menu



Layers context menu shows up when you click with RIGHT mouse button over layer list in the **Layers** tab of **Workspace** pane.

Available options:

- **Add layer** – adds new layer
- **Remove layer** – removes selected layer.

Please note that you can not remove first 5 (built-in) layers

- **Show All** – shows all not locked layers
- **Hide All** – hides all not locked layers
- **Toggle** – toggles visibility of not locked layers
- **Lock built-in layers** – allows you to lock 5 first (built-in) layers. When layer is locked its visibility changes automatically when interval changes and you can not show/hide it manually.
- **Unlock built-in layers** – allows you to unlock 5 first (built-in) layers. Once layer is unlocked its visibility does not change automatically when interval changes and you can show/hide it manually.
- **Properties** – this launches properties box that allows you to rename layer and decide if given layer should or should not be locked to interval displayed. If you mark "Lock visibility to interval" box the layer will show/hide automatically depending on what interval is currently displayed. You can define visibility for **each** layer using "Interval" combo and "Show/hide automatically" buttons. Note that there is a \*separate\* visibility setting for EACH interval. The layer properties box ALWAYS shows "monthly" interval at start but this is just a startup condition you just switch to particular interval and modify visibility. To setup locked layer completely you have to set visibility for **every layer listed** in the "Interval" combo-box. Simply select the interval and choose if layer should be shown or hidden for this interval, select next interval and again choose show or hide, select next and so on...until you define visibility for all intervals.

More information about what layers are and how to use them is in the [Tutorial: Using Layers](#) section of the guide.

#### Real-time quote context menu

Realtime Quote							
Ticker	Open	High	Low	Last	Change	% Change	Volum
AAPL	68.2950	71.4300	68.2000	71.1700	3.9600	+5.89%	60,459,82
ADBE	35.9400	36.1000	35.6000	36.0000	0.0100	+0.03%	1,860,12
ALTR	20.8200	21.2600	20.7500	20.9100	0.0400	+0.19%	2,722,91
AMAT	18.3200	18.6000	18.2600	18.4700	0.2100	+1.15%	12,447,14
AMGN	72.5600	73.1200	71.5700	71.7200	-1.1400	-1.56%	5,411,84
AMZN	37.2000	38.4300	37.1300	38.3500	1.0100	+2.70%	4,251,95
APCC	23.0200			23.0100	0.0300	+0.13%	588,46
APOL	52.9500			52.7100	1.5300	+2.99%	2,546,04
ATYT	16.9400			17.0800	0.1400	+0.83%	5,244,00
BBBY	40.8400			40.7400	2.4200	+6.32%	11,001,66
BEAS	13.3800			12.9800	-0.4200	-3.13%	5,923,80
BIBB	45.3000			44.7775	-0.5825	-1.28%	1,382,83
BMET	37.7500			38.0000	0.2200	+0.58%	1,622,57
BRCM	45.8500			46.2300	0.1760	+0.38%	8,992,08
CDWC	58.7500			58.0630	-0.7070	-1.20%	362,83
CECO	40.8300			40.7200	-0.9800	-2.35%	1,708,99

### Time & Sales

Opens [Time & Sales window](#) that provides information about every bid, ask and trade streaming from the market.

### Easy Alerts

Opens [Easy Alerts](#) window that provides way to define realtime alerts executed when bid/ask/last and other fields hit user-defined levels

### Add Symbol

Adds current symbol to Real-Time Quote list

### Add watch list...

Adds entire watch list to real-time quote window

### Remove Symbol

Removes highlighted line (symbol) from the Real-Time Quote list.

### Remove All

Removes all symbols from real-time quote list

### Hide

Hides Real-Time Quote list

## Keyboard shortcuts

AmiBroker allows complete customization of the user interface, including keyboard shortcuts. To define your own shortcuts use **Tools→Customize** menu, **Keyboard** tab. Read more about it in the [Tutorial: User Interface Customization](#).

Pre-defined keyboard shortcut list follows below, please note that if you used keyboard [customization features](#) the list here may not be valid because some of the entries may have been changed to your own.

Keyboard shortcut	Command
CTRL+0	VIEW_HOURLY
CTRL+1	VIEW_1MINUTE
CTRL+5	VIEW_5MINUTE
CTRL+6	VIEW_15MINUTE
CTRL+C	EDIT_COPY
CTRL+D	VIEW_DAILY
CTRL+E	CHART_EDITFORMULA
CTRL+H	VIEW_CROSSHAIR
CTRL+I	CHART_MORE_INDICATORS
CTRL+M	VIEW_MONTHLY
CTRL+N	FILE_NEW
CTRL+O	FILE_OPEN
CTRL+P	FILE_PRINT
CTRL+R	CHART_PARAMETERS
CTRL+S	FILE_SAVE
CTRL+V	EDIT_PASTE
CTRL+ADD (CTRL+'+')	VIEW_ZOOM_IN
ALT+BACK	EDIT_UNDO
DELETE	EDIT_CLEAR
ALT+DELETE	EDIT_CLEAR_ALL
SHIFT+DELETE	EDIT_CUT
END	CHART_SCROLL_END
F1	HELP
SHIFT+F1	CONTEXT_HELP
F12	CHART_RANGE_BEGIN
CTRL+F12	CHART_RANGE_HIDE
SHIFT+F12	CHART_RANGE_END
F3	STOCK_FIND
F4	QUICK_FIND
F5	VIEW_REFRESH_CHARTS
F6	NEXT_PANE
SHIFT+F6	PREV_PANE
HOME	CHART_SCROLL_BEGIN

CTRL+INSERT	EDIT_COPY
SHIFT+INSERT	EDIT_PASTE
ALT+LEFT	SYMBOL_PREV
CTRL+ALT+LEFT	VIEW_GO_PREV
SHIFT+ALT+LEFT	SYMBOL_PREV_TREE
PAGE_DOWN	CHART_SCROLL_PAGE_RIGHT
CTRL+PAGE_DOWN	VIEW_SHEET_NEXT
PAGE_UP	CHART_SCROLL_PAGE_LEFT
CTRL+PAGE_UP	VIEW_SHEET_PREV
ALT+RETURN	CHART_STUDY_PROPERTIES
ALT+RIGHT	SYMBOL_NEXT
CTRL+ALT+RIGHT	VIEW_GO_NEXT
SHIFT+ALT+RIGHT	SYMBOL_NEXT_TREE
CTRL+SUBTRACT (CTRL+'-')	VIEW_ZOOM_OUT
CTRL+W	VIEW_WEEKLY
CTRL+X	EDIT_CUT
CTRL+Z	EDIT_UNDO

## Import ASCII

AmiBroker has easy-to-use and flexible quotation import feature. This document describes advanced concepts of AmiBroker ASCII importer. Novice users should start with [ASCII Import Wizard](#).

### How does it work?

Quotation data may come from various sources so the format of the ASCII (i.e. text based) file may be much different from one source to another. To handle all those differences AmiBroker uses format definition commands that define the way the text information is interpreted by the ASCII importer. The format definition commands are keywords that begin with a dollar sign '\$'. These commands may be embedded in the data file itself or, may be stored in the separate format definition file for multiple use. Storing format definition commands in separate file avoids the need to include the commands in every data file. The default format definition file name is "default.format". This file, all other ".format" files and "import.types" file (described later) should be stored in **\Formats** subdirectory of AmiBroker's current working directory. The defaults are overridden by any commands included (embedded) in the data file itself.

So, when you use the "Import from ASCII" menu, AmiBroker first looks for the format definition stored in "default.format" file and then parses the file you have chosen. If there is no "default.format" file then it uses internal defaults (described below).

You can modify "default.format" file to suit your needs. Moreover using OLE Automation (Win32 version) or ARexx (Amiga) interface you can specify the name of the format definition file which will be used instead of "default.format" file.

## Format definition commands

The command keywords begin with a dollar sign '\$'. Every line starting with command is interpreted in special way. Here is the list of commands recognized by AmiBroker's built-in importer. Bold letters mark keywords.

<b>Command</b>	<b>\$AUTOADD</b>	Switch new ticker add mode
<b>Arguments</b>	<number>	0 – do not add , 1 – add a new stock when non-existing ticker detected (default = 0)
<b>Alias</b>		
<b>Examples</b>	<b>\$AUTOADD 1</b>	

<b>Command</b>	<b>\$ALLOWNEG</b>	Allow negative numbers in prices
<b>Arguments</b>	<number>	<p>0 – do not allow negative values (default), 1 – allow negative values in prices. This additionally switches off any checking for OHLC relationship so you can import any data into OHLC fields.</p> <p>when \$ALLOWNEG is NOT specified in the ASCII importer definition AmiBroker performs the following range checking and fixup on open, low and high prices  if( open == 0 ) open = close;  if( high &lt; max( open, close ) ) high = max( open, close );  if( low == 0 ) low = min( open, close )</p>
<b>Alias</b>		
<b>Examples</b>	<b>\$ALLOWNEG 1</b>	

<b>Command</b>	<b>\$ALLOW99SECONDS</b>	Convert invalid second stamp
<b>Arguments</b>	<onoff>	<p>This flag works ONLY in conjunction with \$TICKMODE 1 (see below for details)</p> <p>\$ALLOW99SECONDS set to 1 will convert all records with invalid seconds (i.e greater than 59) to 59s. So record stamped 16:29:70 will be treated as 16:29:59</p>
<b>Alias</b>		
<b>Examples</b>	<b>\$ALLOW99SECONDS 1</b>	

<b>Command</b>	<b>\$APPENDNAME</b>	append string to the ticker name (useful when you need to join several fields together to make unique stock symbol)
<b>Arguments</b>	<string>	string to append to the ticker symbol
<b>Alias</b>	<b>\$APPENDTICKER</b>	
<b>Examples</b>		

<b>Command</b>	<b>\$BREAKONERR</b>	Define on-error behaviour
<b>Arguments</b>	<number>	0 – to continue, 1 – to break import on error (default=0)
<b>Alias</b>		
<b>Examples</b>	<b>\$BREAKONERR 1</b>	

<b>Command</b>	<b>\$CONT</b>	Define continuous quotations flag
<b>Arguments</b>	<number>	<0 or 1> – continuous quotations flag, this affects \$AUTOADD 1 mode – if this is set, newly added stocks are switched to continuous quotation mode (this means enabling candlestick charts for example)
<b>Alias</b>		
<b>Examples</b>	<b>\$CONT 1</b>	

<b>Command</b>	<b>\$CURRENCY</b>	Define symbol's currency
<b>Arguments</b>	<string>	Defines currency of symbol
<b>Alias</b>		
<b>Examples</b>	<b>\$CURRENCY EUR</b>  or  <b>\$FORMAT NAME, CURRENCY</b> <b>\$OVERWRITE 1</b> <b>\$AUTOADD 1</b>	



<b>Command</b>	<b>\$DATE_DMY</b>	Define date
<b>Arguments</b>	<number>	The date in Canadian format (DD-MM-YY). If there is no argument given the date is taken from the file name (without an extension)
<b>Alias</b>	<b>\$DATE_CDN</b>	
<b>Examples</b>	<b>\$DATE_DMY 12-05-99</b> <b>\$DATE_CDN 12-05-1999</b>	

<b>Command</b>	<b>\$DATE_MDY</b>	Define date
<b>Arguments</b>	<number>	The date in US format (MM-DD-YY). If there is no argument given the date is taken from the file name (without an extension)
<b>Alias</b>	<b>\$DATE_USA</b>	
<b>Examples</b>	<b>\$DATE_MDY 05/12/99</b> <b>\$DATE_USA 05/12/99</b>	

<b>Command</b>	<b>\$DATE_YMD</b>	Define date
<b>Arguments</b>	<number>	The date in International format (YY-MM-DD). If there is no argument given the date is taken from the file name (without an extension)
<b>Alias</b>	<b>\$DATE_INT</b>	
<b>Examples</b>	<b>\$DATE_INT 99-05-12</b> <b>\$DATE_CDN 1999.05.12</b>	

<b>Command</b>	<b>\$DEBUG</b>	Switch logging (debug) mode
<b>Arguments</b>	<number>	0 – no error logging, 1 – log errors to "import.log" file (default=0)
<b>Alias</b>		
<b>Examples</b>	<b>\$DEBUG 1</b>	

◆

<b>Command</b>	<b>\$FORMAT</b>	Define line format (sequence and types of fields)
<b>Arguments</b>	<b>DATE_MDY</b>	date in US format: MM-DD-YY (alias: <b>DATE_USA</b> )
	<b>DATE_DMY</b>	date in Canadian format: DD-MM-YY (alias: <b>DATE_CDN</b> )
	<b>DATE_YMD</b>	date in International format: YY-MM-DD (alias: <b>DATE_INT</b> )
	<b>TIME</b>	time in HH:MM:SS or HH:MM or HHMM or HHMMSS format
	<b>NAME</b>	ticker name (alias: <b>TICKER</b> )
	<b>ALIAS</b>	symbol alias (\$AUTOADD and \$OVERWRITE modes only)
	<b>FULLNAME</b>	symbol full name (\$AUTOADD and \$OVERWRITE modes only)
	<b>OPEN</b>	open price
	<b>HIGH</b>	high price
	<b>LOW</b>	low price
	<b>CLOSE</b>	close price
	<b>ADJCLOSE</b>	split-adjusted close  This is provided to read adj. close column from Yahoo. Works <b>only</b> in conjunction with CLOSE field. When both CLOSE and ADJCLOSE are present in the ASCII format definition then importer calculates split factor by dividing ADJCLOSE/CLOSE. It then multiplies OPEN, HIGH, LOW and CLOSE fields by this factor and divides VOLUME field by this factor. This effectively converts unadjusted prices to split adjusted prices. Split ratio gets locked once ADJCLOSE drops below

		0.05.
	<b>OPENINT</b>	open interest
	<b>VOLUME</b>	volume
	<b>VOL1000</b>	volume in thousands shares
	<b>VOLMIL</b>	volume in millions shares
	<b>VOLFACTOR</b>	volume factor (number of shares in a block) default =1
	<b>TURNOVER</b>	turnover
	<b>SKIP</b>	skip (ignore) field
	<b>MARKET</b>	specify a field that contains market ID (affects \$AUTOADD and \$OVERWRITE modes only)
	<b>GROUP</b>	specify a field that contains group ID (affects \$AUTOADD and \$OVERWRITE modes only)
	<b>WATCHLIST</b>	specify a field that contains watch list number (0–31) (affects \$AUTOADD and \$OVERWRITE modes only)
	<b>INFO</b>	specify a field with additional information (WSE specific: nk, ns, rk, rs, ok, os, zd, bd )
	<b>REDUCTION</b>	specify a field with reduction rate in percents (WSE specific)
	<b>INDUSTRY</b>	specify a field that contains industry ID (affects \$AUTOADD and \$OVERWRITE modes only)
	<b>APPENDTICKER</b>	specify a field that contains string that should be appended to the ticker name (useful when you need to join several fields together to make unique symbol symbol)
	<b>MARGIN</b>	future contract margin deposit (positive value = dollars, negative value – percent of full value)
	<b>POINTVALUE</b>	future contract point value
	<b>ROUNDLOTSIZE</b>	round lot size (trading unit

		size)
	<b>TICKSIZE</b>	tick size
	<b>ADVISSUES</b>	number of advancing issues
	<b>ADVOLUME</b>	volume of advancing issues
	<b>DECISSUES</b>	number of declining issues
	<b>DECVOLUME</b>	volume of declining issues
	<b>UNCISSUES</b>	number of unchanged issues
	<b>UNCVOLUME</b>	volume of unchanged issues
	<b>CURRENCY</b>	specifies currency of symbol
	<b>DIV_PAY_DATE</b> <b>EX_DIV_DATE</b> <b>LAST_SPLIT_DATE</b> <b>LAST_SPLIT_RATIO</b> <b>EPS</b> <b>EPS_EST_CUR_YEAR</b> <b>EPS_EST_NEXT_YEAR</b> <b>EPS_EST_NEXT_QTR</b> <b>FORWARD_EPS</b> <b>PEG_RATIO</b> <b>BOOK_VALUE</b> (requires <b>SHARES_OUT</b> to be specified as well) <b>BOOK_VALUE_PER_SHARE</b> <b>EBITDA</b> <b>PRICE_TO_SALES</b> (requires <b>CLOSE</b> to be specified as well) <b>PRICE_TO_EARNINGS</b> (requires <b>CLOSE</b> to be specified as well) <b>PRICE_TO_BV</b> (requires <b>CLOSE</b> to be specified as well) <b>FORWARD_PE</b> (requires <b>CLOSE</b> to be specified as well) <b>REVENUE</b> <b>SHARES_SHORT</b> <b>DIVIDEND</b> <b>ONE_YEAR_TARGET</b> <b>MARKET_CAP</b> (requires <b>CLOSE</b> to be specified as well – it is used to calculate shares outstanding) <b>SHARES_FLOAT</b> <b>SHARES_OUT</b> <b>PROFIT_MARGIN</b> <b>OPERATING_MARGIN</b> <b>RETURN_ON_ASSETS</b> <b>RETURN_ON_EQUITY</b>	fundamental data fields. For more info read <a href="#">Using  Fundamental Data</a>

	<b>QTRLY_REVENUE_GROWTH</b> <b>GROSS_PROFIT</b> <b>QTRLY_EARNINGS_GROWTH</b> <b>INSIDER_HOLD_PERCENT</b> <b>INSTIT_HOLD_PERCENT</b> <b>SHARES_SHORT_PREV</b> <b>FORWARD_DIV</b> <b>OPERATING_CASH_FLOW</b> <b>FREE_CASH_FLOW</b> <b>BETA</b>	
<b>Alias</b>		
<b>Examples</b>	<b>\$FORMAT TICKER DATE_MDY OPEN HIGH LOW CLOSE VOLUME</b> <b>\$FORMAT TICKER, DATE_INT, CLOSE, VOLUME</b> <b>\$FORMAT SKIP, TICKER, SKIP, SKIP, DATE_INT, OPEN, HIGH, LOW, CLOSE, TURNOVER</b>	

<b>Command</b>	<b>\$FULLNAME</b>	Define full symbol name
<b>Arguments</b>	<string>	full symbol name
<b>Alias</b>		
<b>Examples</b>	<b>\$FULLNAME Apple Computer Inc.</b>	

<b>Command</b>	<b>\$GROUP</b>	Define group ID
<b>Arguments</b>	<number>	this affects \$AUTOADD 1 mode – if this is specified, newly added symbols are assigned to group with given number.
<b>Alias</b>		
<b>Examples</b>		

<b>Command</b>	<b>\$HYBRID</b>	Switch hybrid mode on/off
<b>Arguments</b>	<number>	0 (off) or 1 (on). When this flag is set, you can combine quotations from multiple files – for example one file can contain only open prices and volume and the other file can contain high/low/close data. Useful especially for Warsaw Stock Exchange for combining the data from fixing and later continuous quotations.
<b>Alias</b>		
<b>Examples</b>		

<b>Command</b>	<b>\$INDUSTRY</b>	Define industry ID
<b>Arguments</b>	<number>	this affects \$AUTOADD 1 mode – if this is specified, newly added symbols are assigned to industry with given number.
<b>Alias</b>		
<b>Examples</b>		

<b>Command</b>	<b>\$MARKET</b>	Define market ID
<b>Arguments</b>	<number>	this affects \$AUTOADD 1 mode – if this is specified, newly added symbols are assigned to market with given number.
<b>Alias</b>		
<b>Examples</b>		

<b>Command</b>	<b>\$NAME</b>	Define ticker name
<b>Arguments</b>	<ticker>	ticker name (symbol) (default = file name without path and extension)
<b>Alias</b>	<b>\$TICKER</b>	
<b>Examples</b>	<b>\$NAME AAPL</b> <b>\$TICKER MSFT</b>	

<b>Command</b>	<b>\$NOQUOTES</b>	Switch quotation data mode
<b>Arguments</b>	<number>	0 – (default) accept only quotation data (AmiBroker checks for non-zero prices and valid dates) 1 – switch off quotation data checking – this allows importing non-quotation data – for example only ticker and full names
<b>Alias</b>	<b>\$TICKER</b>	
<b>Examples</b>	<b>\$NAME AAPL</b> <b>\$TICKER MSFT</b>	

<b>Command</b>	<b>\$OVERWRITE</b>	Switch overwrite mode on/off
<b>Arguments</b>	<number>	0 – off, 1 – on. When overwrite mode is on then information provided by GROUP,

		MARKET, INDUSTRY, FULLNAME fields is overwritten for existing symbols (not only for newly added)
<b>Alias</b>		
<b>Examples</b>	<b>\$OVERWRITE 1</b>	

<b>Command</b>	<b>\$PRICEFACTOR</b>	Define price factor
<b>Arguments</b>	<number>	the factor by which price data are multiplied (default = 1)
<b>Alias</b>		
<b>Examples</b>	<b>\$PRICEFACTOR 100</b>	

<b>Command</b>	<b>\$RAWCLOSE2OI</b>	Put Raw Close price to OI field
<b>Arguments</b>	<number>	0 – off, 1– on. (off by default) – causes that OpenInterest field gets assigned CLOSE (raw close) field value multiplied by 100
<b>Alias</b>		
<b>Examples</b>	<b>\$RAWCLOSE2OI 1</b>	

<b>Command</b>	<b>\$RECALCSPLITS</b>	Recalculate splits
<b>Arguments</b>	<number>	<p>0 – off, 1– on. (off by default) causes that splits are recalculated by AmiBroker by the algorithm that tries to construct correct adjusted price, based on inaccurate information provided by Yahoo.</p> <p>Note that Yahoo provides only 2 decimal digits in adj. close field therefore the more adj. close approaches zero due to adjustments the error grows. The option \$RECALCSPLITS 1 is intended to address this problem (at least partially). It works as follows:</p> <ol style="list-style-type: none"> <li>1. for each bar ratio ADJCLOSE/CLOSE is calculated</li> <li>2. if the ratio changes in two consecutive bars by more than 10% it means that split happened that bar. True split ratio is guessed by matching true fraction</li> </ol>

		in the format of X/Y, where X and Y = 1..9, to the change in ratios. 3. Then true split ratio is used to adjust all past bars until new split is detected.  Works only in conjunction with ADJCLOSE
<b>Alias</b>		
<b>Examples</b>	<b>\$RECALCSPLITS 1</b>	

<b>Command</b>	<b>\$RECALCVOL</b>	Switch automatic index volume recalculation
<b>Arguments</b>	<number>	0 – off, 1 – on (base index only), 2 – on (all indexes). When this is on AmiBroker calculates volumes for indexes based on assignments to markets and base indexes defined in Categories window
<b>Alias</b>		
<b>Examples</b>	<b>\$RECALCVOL 2</b>	

<b>Command</b>	<b>\$RECALCAD</b>	Switch automatic advance/decline composite recalculation
<b>Arguments</b>	<number>	0 – off, 1 – on. When this is on AmiBroker calculates numbers and volumes of issues advancing, declining and unchanged based on assignments to markets and base indexes defined in Categories window.
<b>Alias</b>		
<b>Examples</b>	<b>\$RECALCVOL 2</b>	

<b>Command</b>	<b>\$ROUNDADJ</b>	Round split adjusted prices to given number of decimaldigits
<b>Arguments</b>	<decimaldigits>	<i>decimaldigits</i> – causes split-adjusted prices (see above) to be rounded to 'decimaldigits' precision. By default no rounding is done  Works only in conjunction with



	ADJCLOSE
<b>Alias</b>	
<b>Examples</b>	<b>\$ROUNDADJ 2</b>

<b>Command</b>	<b>\$SEPARATOR</b>	Define field separator character
<b>Arguments</b>	<separator char>	the character used to separate data fields (default = space)
<b>Alias</b>		
<b>Examples</b>	<b>\$SEPARATOR ,</b> <b>\$SEPARATOR ;</b>	

<b>Command</b>	<b>\$SKIPLINES</b>	Define how many lines to skip (ignore)
<b>Arguments</b>	<number>	number of lines to skip (default = 0)
<b>Alias</b>		
<b>Examples</b>	<b>\$SKIPLINES 1</b>	

<b>Command</b>	<b>\$STRICT</b>	Switches on/off strict checking if Open, High, Low prices are greater than zero
<b>Arguments</b>	<onoff>	(default = 0)
<b>Alias</b>		
<b>Examples</b>	<b>\$STRICT 1</b>	

<b>Command</b>	<b>\$TICKMODE</b>	Switches on/off tick mode  \$TICKMODE is a special mode of importer that allows to import quotes that have been already imported.  It makes two assumptions: a) input data should come in the ascending time order (i.e. OLDER records first, LATER records last) b) input data should consist of entire tick history because importer will DELETE any quotes that already exist in the database and then will import new ones.  Once again: Turning on \$TICKMODE 1 will DELETE ANY QUOTES that already exist in the database and then will import new ones. You have been warned.
----------------	-------------------	---

	<p>For example data files like this:</p> <p>MOL,0,20050606,162959,16400.0000,16400.0000,16400.0000,16400.0000,2MOL</p> <p>Can be imported using the following definition file:</p> <pre> \$FORMAT Ticker, Skip, Date_YMD, Time, Open, High, Low, Close, Volume \$SKIPLINES 1 \$SEPARATOR , \$CONT 1 \$GROUP 255 \$AUTOADD 1 \$DEBUG 1 \$TICKMODE 1 </pre> <p>Sometimes it happens that input files have invalid timestamps (seconds &gt; 59).</p> <p>For example:</p> <p>MOL,0,20050606,162970,16400.0000,16400.0000,16400.0000,16400.0000,2</p> <p>Please take a closer look at first line shown in this example it has time:16:29:70 (y</p> <p>So I had to add a special flag to the importer that works around such data errors.</p> <p>It is called \$ALLOW99SECONDS 1 and will convert all records with invalid seconds</p> <p>So record stamped 16:29:70 will be treated as 16:29:59</p> <p>Now for tick mode to work with such incorrect records you would need to add two l</p> <pre> \$TICKMODE 1 \$ALLOW99SECONDS 1 </pre>	
	<b>Arguments</b>	<onoff> (default = 0)
	<b>Alias</b>	
	<b>Examples</b>	<b>\$TICKMODE 1</b>

<b>Command</b>	<b>\$TIMESHIFT</b>	Define intraday time shift used during import
<b>Arguments</b>	<number>	number of hours to shift date/time stamps (can be fractional)
<b>Alias</b>		

<b>Examples</b>	<b>\$TIMESHIFT 2</b> ; will shift 2 hours forward
	<b>\$TIMESHIFT -11.5</b> ; will shift 11 and half hour backward

<b>Command</b>	<b>\$VOLFACTOR</b>	Define volume factor
<b>Arguments</b>	<number>	the factor by which volume data is multiplied (default = 1)
<b>Alias</b>		
<b>Examples</b>	<b>\$VOLFACTOR 10</b>	

<b>Command</b>	<b>\$WATCHLIST</b>	Define watch list number
<b>Arguments</b>	<number>	this affects \$AUTOADD 1 and \$OVERWRITE 1 modes – if this is specified, newly added symbols are added to the watch list with given number.
<b>Alias</b>		
<b>Examples</b>		

#### Notes:

- for **DATE\_xxx** you can use '-', '/' or '\' as day/month/year separators. You can even omit separators at all if only you give a date in a 6 digit (YYMMDD, MMDDYY, DDMMYY) or 8 digit format (YYYYMMDD, MMDDYYYY, DDMMYYYY).
- AmiBroker recognizes decimal as well as true fractions in price data. True fractions must follow the whole value after at least single space. For example you can specify: 5.33 or 5 1/3

AmiBroker is not limited to any kind of fraction, if you wish you can write even: 5 333/999

## Comments

You can include comments in both format definition file and the data file(s). Each line starting with \* (asterisk) or ; (semicolon) or # (hash) is treated as a comment and ignored by the ASCII importer.

## Usage examples

What may look complicated from command list will become quite clear after some examples. So I will give you four examples of how to write format definition files. First example will show the definition for CSV (comma separated values) quotes available from Yahoo's finances site. Second example will show definition for Metastock ASCII file format. Third example shows definition for Omega SuperCharts ASCII file format. And fourth example will show the definition for s-files used by DM BOS (Polish brokerage company).

**Yahoo CSV**

The data from Yahoo's site looks as follows:

```
Date,Open,High,Low,Close,Volume
1-Feb- 0,104,105,100,100.25,2839600
31-Jan- 0,101,103.875,94.50,103.75,6265000
28-Jan- 0,108.1875,110.875,100.625,101.625,3779900
```

The first line gives us a hint about the meaning of the comma separated fields. First field will hold the date. The remaining fields will hold open, high, low, close prices and volume. Importer should skip the first line and parse all the remaining lines that hold just comma-separated data. Appropriate format definition file would look like this:

```
$FORMAT Date_DMY,Open,High,Low,Close,Volume
$SKIPLINES 1
$SEPARATOR ,
$DEBUG 1
$AUTOADD 1
$BREAKONERR 1
```

\$DEBUG switches on error logging to "import.log" file and \$BREAKONERR will cause importer to stop after the first error found. \$AUTOADD ensures that new ticker will be added to the database if it is missing. Well... you may ask: how does it know the ticker name? The answer is simple: if there is no field which defines the ticker name, the importer takes the file name (without path and extension) as a ticker. So if you are importing file "C:\My data\AAPL.CSV" AmiBroker will use "AAPL" as a ticker name.

**Metastock ASCII**

The data in Metastock ASCII format looks as follows:

```
<ticker>,<per>,<date>,<high>,<low>,<close>,<vol>
AAP,D,1/17/2000,5483.33,5332.01,5362.3,0
AKS,D,1/17/2000,9868.45,9638.03,9687.62,0
FET,D,1/17/2000,3741.3,3540.2,3570.81,0
```

First field will hold the ticker name, second – time period ("D" means daily data), third – quotation date. The rest will hold high, low, close prices and volume. The importer should then skip the first line and parse all the remaining lines that hold just comma-separated data. Appropriate format definition file would look like this:

```
$FORMAT Ticker,Skip,Date_MDY,High,Low,Close,Volume
$SKIPLINES 1
$SEPARATOR ,
$DEBUG 1
$AUTOADD 1
$BREAKONERR 1
```

Skip in \$FORMAT defines a field which should be ignored by the importer.

**Omega SuperCharts ASCII**

The data in Omega SC ASCII format looks as follows:

```
ticker,date,open,high,low,close,vol
AAP,20000117,5333.01,5483.33,5332.01,5362.3,3433450
```

This format is similar to previous ones, however the date is in YYYYMMDD format without separators between year, month and day part. AmiBroker, however, can handle such dates with ease. Appropriate format definition file would look like this:

```
$FORMAT Name,Date_Int,Open,High,Low,Close,Volume
$SEPARATOR ,
$DEBUG 1
$SKIPLINES 1
$AUTOADD 1
$BREAKONERR 1
```

Skip in \$FORMAT defines a field which should be ignored by the importer.

**DMBOS S-files**

The data in this format looks as follows:

```
0,29-02-00,12:05,MIDWIG,1069.1,,,+1.2,336002000,
0,29-02-00,12:05,NIF,48.6,,,+0.8,1763000,
0,29-02-00,12:05,WIG20,2300.3,,,+1.1,336002000,
0,29-02-00,12:05,WIG,21536.8,,,+0.2,336002000,
0,29-02-00,12:05,WIRR,2732.8,,,+1.6,16373000,
1,29-02-00,12:05,AGORA,144.00,,,+4.7,15802000,
1,29-02-00,12:05,AGROS,40.00,nk,72,+5.0,840000,
1,29-02-00,12:05,AMERBANK,28.00,,,+3.7,22000,
1,29-02-00,12:05,AMICA,41.50,nk,99,+2.2,564000,
```

This format is a little bit more complicated. For us useful fields are: 2nd – date, 4th – ticker, 5th – close price, 9th – the turnover value (close \* volume). The remaining fields holds other information that is not useful for us. Appropriate format definition file would look like this:

```
$FORMAT Skip,Date_DMY,Skip,Name,Close,Skip,Skip,Skip,Turnover
$SEPARATOR ,
$DEBUG 1
```

**Default behaviour**

When importing ASCII files, AmiBroker attempts to open "default.format" file (in the AmiBroker's directory) to obtain the format definition. If such file is missing the following default format is applied:

```
$FORMAT DATE_USA, OPEN, HIGH, LOW, CLOSE, VOLUME
$SEPARATOR
```

This means that by default ASCII importer will use space character as a separator and will parse the following

fields: date, open, high, low, close, volume. The file name (without path and extension) will be used as a ticker name. All other import parameters (\$DEBUG,\$AUTOADD, etc.) are set to zero.

## User-definable file types and formats

Now AmiBroker can use not only default.format definition file but also other user-specified files. File types, filters and format definition files are specified in **import.types** file (example is included in the update package). Now user can prepare/modify **import.types** file with the description of supported ASCII formats and filters to use. The format of **import.types** file is:

```
<Descriptive name>|<File filter>|<definition file name>
```

Note vertical line characters between these three fields. Example import.types file looks as follows:

```
Default ASCII (*.*)|*.*|default.format
Yahoo's CSV (*.csv)|*.csv|yahoo.format
Metastock ASCII (*.mst)|*.mst|metastock.format
Omega SC ASCII (*.txt)|*.txt|omega.format
S-Files (s*.*)|s*.*|sfile.format
C-Files (c*.*)|c*.*|cfile.format
Sharenet DAT (*.dat)|*.dat|dat.format
```

If such file exists you will see your types in the "Files of type" combo-box and when you select one – appropriate filter will be used and after selecting some files and clicking OK – importer will use specified ".format" file.

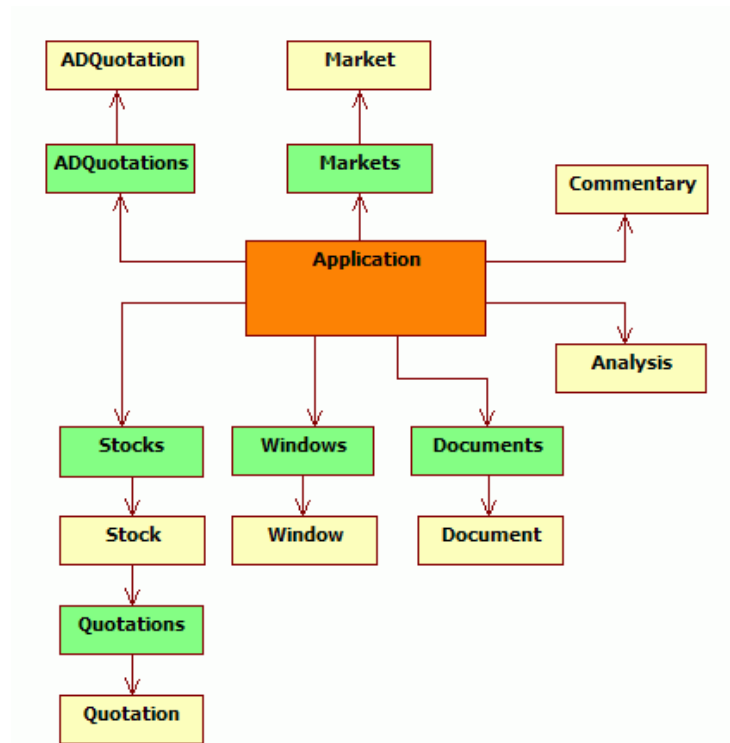
In that way you can define as many text-based data formats as you like and AmiBroker will be able to "understand" them all.

## Ticker aliases

Now each ticker can have an alias assigned, so the AmiBroker's built-in importers can recognize that security by both ticker symbol and alias names. This is useful when you are using two data sources that are using slightly different symbol naming convention or if you want to give the symbols more intuitive name while retaining the ability to use importers without problems.

## AmiBroker's OLE Automation Object Model

AmiBroker object model hierarchy. V5.00



## Index of objects

- ADQuotation
- ADQuotations
- Analysis
- Application
- Window
- Windows
- Commentary
- Document
- Documents
- Market
- Markets
- Quotation
- Quotations
- Stock
- Stocks

## ADQuotation

### Properties:

- ◆ **Date** As Date
- ◆ **AdvIssues** As Long
- ◆ **AdvVolume** As Single
- ◆ **DeclIssues** As Long
- ◆ **DeclVolume** As Single
- ◆ **UncIssues** As Long
- ◆ **UncVolume** As Single

### Description:

ADQuotation class keeps one bar of advance/decline information

## ADQuotations

**Methods:**

- ◆ Function **Add**(ByVal **Date** As Variant) As Object
- ◆ Function **Remove**(ByVal **Date** As Variant) As Boolean

**Properties:**

- ◆ **Item**(ByVal **Date** As Variant) As Object [r/o] [default]
- ◆ **Count** As Long

**Description:**

ADQuotations is a collection of ADQuotation objects

**Analysis****Properties:**

- ◆ Property **Filter**(ByVal **nType** As Integer, ByVal **pszCategory** As String) As Long [r/w]

**Methods:**

- ◆ Sub **Backtest**([ByVal **Type** As Variant])
- ◆ Sub **ClearFilters**()
- ◆ Sub **Edit**([ByVal **bForceReload** As Variant])
- ◆ Sub **Explore**()
- ◆ Function **Export**(ByVal **pszFileName** As String) As Boolean
- ◆ Function **LoadFormula**(ByVal **FileName** As String) As Boolean
- ◆ Function **LoadSettings**(ByVal **pszFileName** As String) As Boolean
- ◆ Sub **MoveWindow**(ByVal **Left** As Long, ByVal **Top** As Long, ByVal **Width** As Long, ByVal **Height** As Long)
- ◆ Sub **Optimize**([ByVal **Type** As Variant])
- ◆ Function **Report**(ByVal **pszFileName** As String) As Boolean
- ◆ Function **SaveFormula**(ByVal **pszFileName** As String) As Boolean
- ◆ Function **SaveSettings**(ByVal **pszFileName** As String) As Boolean
- ◆ Sub **Scan**()
- ◆ Sub **ShowWindow**(ByVal **nShowCmd** As Long)
- ◆ Sub **SortByColumn**(ByVal **iColumn** As Long, ByVal **bAscending** As Integer, ByVal **bMultiMode** As Integer)

**Properties:**

- ◆ **RangeMode** As Long
- ◆ **RangeN** As Long
- ◆ **RangeFromDate** As Date
- ◆ **RangeToDate** As Date
- ◆ **ApplyTo** As Long

**Description:**

Analysis object provides programmatic control of automatic analysis window

**Notes:**

Analysis.Backtest( Type = 2 ); – runs backtest



Type parameter can be one of the following values:

- 0 : portfolio backtest/optimize
- 1 : individual backtest/optimize
- 2 : old backtest/optimize

Analysis.Optimize(Type = 2 ); – runs optimization

Type parameter can be one of the following values:

- 0 : portfolio backtest/optimize
- 1 : individual backtest/optimize
- 2 : old backtest/optimize

Analysis.Report( FileName: String ) – saves report to the file or displays it if FileName = ""

Analysis.ApplyTo – defines apply to mode: 0 – all stocks, 1 – current stock, 2 – use filter

Analysis.RangeMode – defines range mode: 0 – all quotes, 1 – n last quotes, 2 – n last days, 3 – from-to date

Analysis.RangeN – defines N (number of bars/days to backtest)

Analysis.RangeFromDate – defines "From" date

Analysis.RangeToDate – defines "To" date

Analysis.Filter( nType: short, Category : String ) – sets/retrieves filter setting

nType argument defines type of filter 0 – include, 1 – exclude

Category argument defines filter category:

"index", "favorite", "market", "group", "sector", "index", "watchlist"

## Application

### Methods:

- ◆ Function **Import**(ByVal **Type** As Integer, ByVal **FileName** As String, [ByVal **DefFileName** As Variant]) As Long
- ◆ Function **LoadDatabase**(ByVal **Path** As String) As Boolean
- ◆ Function **LoadLayout**(ByVal **pszFileName** As String) As Boolean
- ◆ Function **Log**(ByVal **Action** As Integer) As Long
- ◆ Sub **Quit**()
- ◆ Sub **RefreshAll**()
- ◆ Sub **SaveDatabase**()
- ◆ Function **SaveLayout**(ByVal **pszFileName** As String) As Boolean

### Properties:

- ◆ **ActiveDocument** As Object
- ◆ **Stocks** As Object
- ◆ **Version** As String
- ◆ **Documents** As Object
- ◆ **Markets** As Object
- ◆ **DatabasePath** As String
- ◆ **Analysis** As Object
- ◆ **Commentary** As Object
- ◆ **ActiveWindow** As Object
- ◆ **Visible** As Integer

**Description:**

Application object is main OLE automation object for AmiBroker. You have to create it prior to accessing any other objects. To create Application object use the following code:

JScript:

```
AB = new ActiveXObject("Broker.Application");
```

VB/VBScript:

```
AB = CreateObject("Broker.Application")
```

AFL:

```
AB = CreateObject("Broker.Application");
```

**Window****Methods:**

- ◆ Sub **Activate**()
- ◆ Sub **Close**()
- ◆ Function **ExportImage**(ByVal **FileName** As String, [ByVal **Width** As Variant], [ByVal **Height** As Variant], [ByVal **Depth** As Variant]) As Boolean
- ◆ Function **LoadTemplate**(ByVal **IpszFileName** As String) As Boolean
- ◆ Function **SaveTemplate**(ByVal **IpszFileName** As String) As Boolean
- ◆ Function **ZoomToRange**(ByVal **From** As Variant, ByVal **To** As Variant) As Boolean

**Properties:**

- ◆ **SelectedTab** As Long
- ◆ **Document** As Object

**Description:**

Window object provides programmatic control over charting window.

**Windows****Methods:**

- ◆ Function **Add**() As Object

**Properties:**

- ◆ **Item**(ByVal **Index As Long**) As Object [r/o] [default]
- ◆ **Count** As Long

**Description:**

Windows is a collection of Window objects.

## Commentary

### Methods:

- ◆ Sub **Apply**()
- ◆ Sub **Close**()
- ◆ Function **LoadFormula**(ByVal **pszFileName** As String) As Boolean
- ◆ Function **Save**(ByVal **pszFileName** As String) As Boolean
- ◆ Function **SaveFormula**(ByVal **pszFileName** As String) As Boolean

### Description:

Commentary object gives programmatic control over guru commentary window.

## Document

### Methods:

- ◆ Sub **Activate**()
- ◆ Sub **Close**()
- ◆ Sub **ShowMessage**(ByVal **Text** As String)

### Properties:

- ◆ **Application** As Object
- ◆ **Parent** As Object
- ◆ **Name** As String
- ◆ **ActiveWindow** As Object
- ◆ **Windows** As Object

### Description:

Document object represents active document (of 'chart' type). In document–view architecture each document can have multiple windows (views) connected. Name property defines currently selected symbol for the document.

## Documents

### Methods:

- ◆ Function **Add**() As Object
- ◆ Sub **Close**()
- ◆ Function **Open**(ByVal **Ticker** As String) As Object

### Properties:

- ◆ **Item**(ByVal **Index** As Long) As Object [r/o] [default]
- ◆ **Count** As Long
- ◆ **Application** As Object
- ◆ **Parent** As Object

### Description:

Documents is a collection of document objects.

## Market

### Properties:

- ◆ **Name** As String
- ◆ **ADQuotations** As Object

### Description:

Market represents market category and its related data (i.e. per-market advance/decline information)

## Markets

### Properties:

- ◆ **Item**(ByVal **Index As Integer**) As Object [r/o] [default]
- ◆ **Count** As Integer

### Description:

Markets is a collection of Market objects

## Quotation

### Properties:

- ◆ **Date** As Date
- ◆ **Close** As Single
- ◆ **Open** As Single
- ◆ **High** As Single
- ◆ **Low** As Single
- ◆ **Volume** As Single
- ◆ **OpenInt** As Single

### Description:

Quotation class represents one bar of price data

## Quotations

### Methods:

- ◆ Function **Add**(ByVal **Date** As Date) As Object
- ◆ Function **Remove**(ByVal **Item** As Variant) As Boolean
- ◆ Function **Retrieve**(ByVal **Count** As Long, ByRef **Date** As Variant, ByRef **Open** As Variant, ByRef **High** As Variant, ByRef **Low** As Variant, ByRef **Close** As Variant, ByRef **Volume** As Variant, ByRef **OpenInt** As Variant) As Long

### Properties:

- ◆ **Item**(ByVal **Item As Variant**) As Object [r/o] [default]
- ◆ **Count** As Long

**Description:**

Quotations is a collection of Quotation objects. It represents all quotations available for given symbol. Quotations collection is available as a property of Stock object.

**Stock****Properties:**

- ◆ **Ticker** As String
- ◆ **Quotations** As Object
- ◆ **FullName** As String
- ◆ **Index** As Boolean
- ◆ **Favourite** As Boolean
- ◆ **Continuous** As Boolean
- ◆ **MarketID** As Long
- ◆ **GroupID** As Long
- ◆ **Beta** As Single
- ◆ **SharesOut** As Single
- ◆ **BookValuePerShare** As Single
- ◆ **SharesFloat** As Single
- ◆ **Address** As String
- ◆ **WebID** As String
- ◆ **Alias** As String
- ◆ **IsDirty** As Boolean
- ◆ **IndustryID** As Long
- ◆ **WatchListBits** As Long
- ◆ **DataSource** As Long
- ◆ **DataLocalMode** As Long
- ◆ **PointValue** As Single
- ◆ **MarginDeposit** As Single
- ◆ **RoundLotSize** As Single
- ◆ **TickSize** As Single
- ◆ **WatchListBits2** As Long
- ◆ **Currency** As String
- ◆ **LastSplitFactor** As String
- ◆ **LastSplitDate** As Date
- ◆ **DividendPerShare** As Single
- ◆ **DividendPayDate** As Date
- ◆ **ExDividendDate** As Date
- ◆ **PEGRatio** As Single
- ◆ **ProfitMargin** As Single
- ◆ **OperatingMargin** As Single
- ◆ **OneYearTargetPrice** As Single
- ◆ **ReturnOnAssets** As Single
- ◆ **ReturnOnEquity** As Single
- ◆ **QtrlyRevenueGrowth** As Single
- ◆ **GrossProfitPerShare** As Single
- ◆ **SalesPerShare** As Single
- ◆ **EBITDAPerShare** As Single
- ◆ **QtrlyEarningsGrowth** As Single

- ◆ **InsiderHoldPercent** As Single
- ◆ **InstitutionHoldPercent** As Single
- ◆ **SharesShort** As Single
- ◆ **SharesShortPrevMonth** As Single
- ◆ **ForwardDividendPerShare** As Single
- ◆ **ForwardEPS** As Single
- ◆ **EPS** As Single
- ◆ **EPSEstCurrentYear** As Single
- ◆ **EPSEstNextYear** As Single
- ◆ **EPSEstNextQuarter** As Single
- ◆ **OperatingCashFlow** As Single
- ◆ **LeveredFreeCashFlow** As Single

**Description:**

Stock class represents single symbol data. For historical reasons the name of the object is Stock, but it can hold any kind of instrument (including futures, forex, etc).

**Stocks****Methods:**

- ◆ Function **Add**(ByVal **Ticker** As String) As Object
- ◆ Function **GetTickerList**(ByVal **nType** As Long) As String
- ◆ Function **Remove**(ByVal **Item** As Variant) As Boolean

**Properties:**

- ◆ **Item**(ByVal **Item** As Variant) As Object [r/o] [default]
- ◆ **Count** As Long

**Description:**

Stocks is a collection of Stock objects. It is available as a property of Application object.

**Notes:**

Stock.WatchListBits (long) – each bit 0..31 represents assignment to one of 32 watch lists to add a stock to nth watch list write (JScript example):

```
Stock.WatchListBits |= 1 << nth;
```

Stock.WatchListBits2 (long) – each bit 0..31 represents assignment to one of watch lists numbered from 32..63 to add a stock to nth watch list write (JScript example):

```
Stock.WatchListBits2 |= 1 << ( nth – 32 );
```

Stock.DataSource ( 0 – default, 1 – local only )

Stock.DataLocalMode ( 0 – default, 1 – store locally, 2 – don't store locally)

**Practical Examples:****Example 1: Running simple backtest**

```
/* create AB object */
AB = new ActiveXObject( "Broker.Application" );
```

```

/* retrieve automatic analysis object */
AA = AB.Analysis;

/* load formula from external file */
AA.LoadFormula("afl\\macd_c.afl");

/* optional: load settings */
// AA.LoadSettings("the_path_to_the_settings_file.abs");

/* setup filters */
/* backtest over symbols present in market 0 only (zero-based number)
*/
AA.ClearFilters();
AA.Filter( 0, "market" ) = 0;

// uncomment line below to
// AA.Filter( 1, "market" ) = 2; // exclude 2nd market

/* set apply to and range */
AA.ApplyTo = 2; // use filters
AA.RangeMode = 0; // use all available quotes

/* run backtest and display report */
AA.Backtest();
AA.Report(""); // empty file name means display report

```

### Example 2: Batch backtesting

Caution: It will run backtest of EVERY formula stored in C:\Program Files\AmiBroker\AFL on all symbols of current database. After each backtest the report is generated and saved into the file named <formula name>.HTML.

You can modify this AFL path in the script itself (you can open it with Notepad).

Below comes the listing.

```

/*****
*
* BatchTest.js
*
* Batch testing sample script
* Shows how to use JScript and new AmiBroker 4.23
* 'Analysis' object to perform batch backtesting
* and generate reports
*
* Created: Dec 21, 2002 TJ
* Last modification: Dec 22, 2002 TJ
*
* Copyright (C)2002 Amibroker.com

```

```

*
* Status: Freeware
* You can use/modify/adopt this code freely
*
*/

/* The directory where AFL files are stored
** Also reports generated by the bactest
** will be saved here
*/

AFLFolder = "C:\\Program Files\\AmiBroker\\AFL"; // MODIFY TO FIT YOUR SETUP

WScript.Echo("Batch testing of all AFL files stored in " + AFLFolder );

var AB, AA;
var fso, f, fl, fc, s;
var filename;

/* Create AmiBroker object and get Analysis object */
AB = new ActiveXObject("Broker.Application");
AA = AB.Analysis;

/* backtest over symbols and all quotes*/
AA.ClearFilters();
AA.ApplyTo = 0; // use symbols
AA.RangeMode = 0; // all quotes

/* to use filters you should uncomment lines below
// AA.ApplyTo = 2; // use filters
// AA.Filter(0,"watchlist") = 2 /* watch list number */;
// AA.Filter(0,"group") = 0 /* group number */;

/* Create FileSystemObject */
fso = new ActiveXObject("Scripting.FileSystemObject");

/* Iterate through all files in the folder */
f = fso.GetFolder(AFLFolder);
fc = new Enumerator(f.files);
for (; !fc.atEnd(); fc.moveNext())
{
    // we need to add empty string to make sure that filename is a string object
    filename = "" + fc.item();

    /* check the AFL extension */
    if( filename.substr( filename.length - 4 ).toUpperCase() == ".AFL" )
    {
        if( AA.LoadFormula( filename ) )
        {
            AA.Backtest();
        }
    }
}

```



```

        reportname = filename.substr( 0, filename.length - 3 ) + "HTML" ;

        AA.Report( reportname ); // generate report
    }
}

```

### Example 3: Execute commentary

```

AB = new ActiveXObject("Broker.Application");
AB.Commentary.LoadFormula("C:\\Program Files\\AmiBroker\\AFL\\MACD_c.afl");
AB.Commentary.Apply();
AB.Commentary.Save("Test.txt");
AB.Commentary.SaveFormula("MACDTest.afl");
//AB.Commentary.Close();

```

## AmiQuote's OLE Automation Object Model

### Index of objects

- [Document](#)

AmiQuote is SDI (single document) application therefore there is only one class – Document – creatable using the following code:

JScript:

```
AB = new ActiveXObject("AmiQuote.Document");
```

VB/VBScript:

```
AB = CreateObject("AmiQuote.Document")
```

AFL:

```
AB = CreateObject("AmiQuote.Document");
```

## Document

### Methods:

- ◆ Function **AddSymbols**(ByVal **pszSymbols** As String) As Boolean
- ◆ Function **Download**() As Boolean
- ◆ Function **GetSymbolsFromAmiBroker**() As Boolean
- ◆ Function **Import**() As Boolean
- ◆ Sub **MoveWindow**(ByVal **x** As Long, ByVal **y** As Long, ByVal **width** As Long, ByVal **height** As Long)
- ◆ Function **Open**(ByVal **pszFileName** As String) As Boolean
- ◆ Function **RemoveAllSymbols**() As Boolean
- ◆ Function **RemoveSymbols**(ByVal **pszSymbols** As String) As Boolean
- ◆ Function **Save**() As Boolean
- ◆ Function **SaveAs**(ByVal **pszFileName** As String) As Boolean

### Properties:

- ◆ **DownloadInProgress** As Boolean
- ◆ **ImportInProgress** As Boolean
- ◆ **Source** As Long
- ◆ **From** As Date
- ◆ **To** As Date
- ◆ **AutoImport** As Boolean
- ◆ **AllSessions** As Boolean
- ◆ **Interval** As Long
- ◆ **RunEvery** As Long
- ◆ **DestinationFolder** As String

# Technical analysis guide

[Introduction](#)

[Basic tools](#)

[Indicators](#)

## Introduction

Technical analysis is the examination of past price movements to forecast future price movements. Technical analysts are sometimes referred to as chartists because they rely almost exclusively on charts for their analysis.

Technical analysis is applicable to stocks, indices, commodities, futures, currencies or any tradable instrument where the price is influenced by the forces of supply and demand. Price refers to any combination of the open, high, low or close for a given security over a specific timeframe. The time frame can be based on intraday, daily, weekly or monthly price data and last a few hours or many years. In addition, some technical analysts include volume or open interest figures with their study of price action.

AmiBroker provides a comprehensive set of technical analysis tools that will be presented in this chapter.

## Basic tools

AmiBroker has following basic technical analysis tools:

- [Price charts](#)
- [Trend lines](#)
- [Moving averages](#)
- [Fibonacci retracement](#)
- [Fibonacci time zones](#)
- [Regression channels](#)
- [Bollinger bands](#)

## Price charts

AmiBroker can display the prices using:

- **line chart**  
this mode is used when current symbol uses price fixing and only close price is available
- **traditional bar chart**  
this mode is used when continuous trading is enabled, but open price is not available (or equals to close price)
- **Japanese Candlesticks**  
this mode is used when continuous trading is enabled with open/close/high/low data

A line chart is the simplest type of chart. One price (close) is plotted for each time period. A single line connects each of these price points. The main strength of this chart type is simplicity.

Bar charts are one of the most popular types of charts used in technical analysis. For each trading day a vertical line is plotted. The top of the vertical line indicates the highest price a security traded at during the day, and the bottom represents the lowest price. The closing price is displayed by the mark on the right side of

the bar and opening prices are shown on the left side of the bar.

Developed by the Japanese in the 1600's, candlestick charts are merely bar charts that extenuate the relationship between open, high, low and closing prices. Each candlestick represents one period of data (day–week) and consists of an upper shadow, lower shadow and the body. The upper shadow is the highest price that the stock traded at for the period while the lower shadow represents the lowest price. The candlestick body is black when the close is less than the open or white when the close is greater than the open. The top of the body is the opening price if the candle is black and the candle is referred to as a long black candle. If the candle is white, the top of the body is the closing price and the candle is referred to as a long white candle.

Steven Nison's articles that explain Candlestick charting appeared in the December, 1989 and April, 1990 issues of Futures Magazine. The definitive book on the subject is Japanese Candlestick Charting Techniques also by Steve Nison.

There are many different candlestick formations. Some are considered to be minor formations while others are major. Candlestick charts dramatically illustrate supply/demand concepts defined by classical technical analysis theories.

### Major Candlestick Chart Formations:

**Gravestone Doji:** A doji (open and close are the same) and the high is significantly higher than the open, high and closing prices. This formation typically occurs at the bottom of a trend and signals a bullish reversal.

**Dragon–fly Doji:** A doji (open and close are the same) and the low is significantly lower than the open, high and closing prices. This formation typically occurs at the top of a trend and signals a bearish reversal.

**Abandoned Baby Doji:** A doji, which occurs at the bottom of a chart formation with gaps on both sides of the doji.

**Harami Cross:** This formation signals a market top. It consists of a harami, which is a long black line candlestick which precedes and engulfs a doji with no body.

**Engulfing Pattern:** A two–candle bullish formation consisting of a small long black line engulfed by the second candle, a long white line.

**Evening Star:** A bearish pattern usually occurring at a top. The formation consists of three candles. The first is a long white line followed by a star and then a long black line. The star can be either black or white.

**Dark Cloud Cover:** A two candle formation whereby the first candle is a long white line and the second candle is a long black line whose body is below the center of the first candle. This is a bearish formation.

## Trend lines

Technical analysis is built on the assumption that prices trend. Trendlines are an important tool in technical analysis for both trend identification and confirmation. A trendline is a straight line that connects two or more price points and then extends into the future to act as a line of support or resistance. Many of the principles applicable to support and resistance levels can be applied to trendlines as well.

### Up Trendline

An up trendline has a positive slope and is formed by connecting two or more low points. The second low must be higher than the first for the line to have a positive slope. Up trendlines act as support and indicate that net-demand (demand less supply) is increasing even as the price rises. A rising price combined with increasing demand is very bullish and shows a strong determination on the part of the buyers. As long as prices remain above the trendline, the uptrend is considered solid and intact. A break below the up trendline indicates that net-demand has weakened and a change in trend could be imminent.

### **Down Trendline**

A down trendline has a negative slope and is formed by connecting two or more high points. The second high must be lower than the first for the line to have a negative slope. Down trendlines act as resistance and indicate that net-supply (supply less demand) is increasing even as the price declines. A declining price combined with increasing supply is very bearish and shows the strong resolve of the sellers. As long as prices remain below the down trendline, the downtrend is considered solid and intact. A break above the down trendline indicates that net-supply is decreasing and a change of trend could be imminent.

### **Scale Settings**

High points and low points appear to line up better for trendlines when prices are displayed using a semi-log scale. This is especially true when long-term trendlines are being drawn or there has been a large change in price. AmiBroker allows to set the scale as arithmetic or logarithmic (semi-log). An arithmetic scale displays incremental values (5,10,15,20,25,30) evenly as they move up the y-axis. A \$10 movement in price will look the same from \$10 to \$20 or from \$100 to \$110. A semi-log scale displays incremental values in percentage terms as they move up the y-axis. A move from \$10 to \$20 is a 100% gain and would appear to be a much larger than a move from \$100 to \$110, which is only a 10% gain.

Please remember however that straight line in the log chart is no longer straight in the linear scale, so trend lines drawn in one scale may look strange in the other scale.

### **Validation**

It takes two or more points to draw a trendline. The more points used to draw the trendline, the more validity attached to the support or resistance level represented by the trendline. It can sometimes be difficult to find more than 2 points from which to construct a trendline. Even though trendlines are an important aspect of technical analysis, it is not always possible to draw trendlines on every price chart. Sometimes the lows or highs just don't match up and it is best not to force the issue. The general rule in technical analysis is that it takes two points to draw a trendline and the third point confirms the validity.

## **Moving averages**

The moving average is one of the most useful, objective and oldest analytical tools around. Some patterns and indicators can be somewhat subjective, where analysts may disagree on if the pattern is truly forming or if there is a deviation that is might be an illusion. The moving average is more of a cut-and-dry approach to analyzing stock charts and predicting performance, and it is one of the few that doesn't require a genius intelligence to interpret..

Moving average is an indicator that shows the average value of a security's price over a period of time.

To find the 50 day Simple Moving Average you would add up the closing prices (but not always more later)

from the past 50 days and divide them by 50. And because prices are constantly changing it means the moving average will move as well.

Exponential Moving Average (EMA) – is calculated by applying a percentage of today's closing price to yesterday's moving average value. Use an exponential moving average to place more weight on recent prices. As expected, each new price has a greater impact on the EMA than it has on the SMA. And, each new price changes the moving average only once, not twice.

The most commonly used moving averages are the 15, 20, 30, 45, 50, 100, and 200 day averages. Each moving average provides a different interpretation on what the stock price will do. There really isn't just one "right" time frame. Moving averages with different time spans each tell a different story. The shorter the time span, the more sensitive the moving average will be to price changes. The longer the time span, the less sensitive or the more smoothed the moving average will be. Moving averages are used to emphasize the direction of a trend and smooth out price and volume fluctuations or "noise" that can confuse interpretation.

Different investors use moving averages for different reasons. While some use it as their primary analytic tool others simply use the moving average as confidence builder to back their investment decisions. Here are two other strategies that people use moving averages for:

### **Filters**

Filtering is used to increase your confidence about an indicator. There are no set rules or things to look out for when filtering, just whatever makes you confident enough to invest your money. For example you might want to wait until a security crosses through its moving average and is at least 10% above the average to make sure that it is a true crossover. Remember, setting the percentile too high could result in "missing the boat" and buying the stock at its peak.

Another filter is to wait a day or two after the security crosses over, this can be used to make sure that the rise in the security isn't a fluke or unsustained. Again, the downside is if you wait too long then you could end up missing some big profits.

### **Crossovers**

Using Crossovers isn't quite as easy as filtering. There are several different types of crossover's, but all of them involve two or more moving averages. In a double crossover you are looking for a situation where the shortest MA crosses through the longer one. This is almost always considered to be a buying signal since the longer average is somewhat of a support level for the stock price.

For extra insurance you can use a triple crossover, whereby the shortest moving average must pass through the two higher ones. This is considered to be an even stronger buying indicator.

### **Regression channels**

Linear regression may sound intimidating, but the mathematical concept is a simple one. All this technique does is fit a straight line through a finite number of data points by minimizing the sum of the squared vertical distance between the line and each of the points. In our context, this means that if time is represented by days on the horizontal axis and the closing price on those days is plotted as dots on the vertical axis (a normal closing price chart), then we try to fit a straight line through those closing-price dots such that the total sum of the squared vertical distance between each closing price and the line are minimized. This would then be our best-fit line.

Raff regression channel Raff Regression Channels show the range prices can be expected to deviate from a Linear Regression trend line. Developed by Gilbert Raff, the regression channel is a line study the plots directly on the price chart. The Regression Channel provides a precise quantitative way to define a price trend and its boundaries. The Regression Channel is constructed by plotting two parallel, equidistant lines above and below a Linear Regression trend line.

The distance between the channel lines to the regression line is the greatest distance that any one high or low price is from the regression line.

Raff Regression Channels contain price movement, with the bottom channel line providing support and the top channel line providing resistance. Prices may extend outside of the channel for a short period of time. However, if prices remain outside the channel for a long period of time, a reversal in trend may be imminent.

## Fibonacci Retracement

Fibonacci Retracements/Extensions are displayed by first drawing a trendline between two extreme points. After selecting Fibonacci Retracement tool from **Draw** toolbar, a series of up to nine horizontal lines will be drawn at the Fibonacci levels of 0.0%, 23.6%, 38.2%, 50.0%, 61.8%, 100%, 161.8%, 261.8% and 423.6%. After a significant move (up or down), prices will often rebound and retrace a significant portion of the original move. As the price retraces, support and resistance levels will often occur near the Fibonacci Retracement levels.

Fibonacci retracement/extension tool works in 4 different modes depending on the direction of trend line drawn:

- NE – gives (old-style) retracement in up trend
- SE – gives retracement in down trend
- NW – gives extension in up trend
- SW – gives extension in down trend

A controlling trend line drawn with dotted style can be used to delete Fibonacci retracement study at once using right mouse button menu.

## Fibonacci Time Zones

The Fibonacci Time Zones study consists of vertical lines at the Fibonacci intervals of 1, 2, 3, 5, 8, 13, 21, 34, etc. The interpretation of Fibonacci Time Zones involves looking for significant changes in price near the vertical lines.

## Bollinger bands

Bollinger Bands are envelopes which surround the price bars on a chart. Bollinger Bands are plotted two standard deviations away from a simple short-term moving average. This is the primary difference between Bollinger Bands and envelopes. Envelopes are plotted a fixed percentage above and below a moving average. Because standard deviation is a measure of volatility, the Bollinger Bands adjust themselves to the market conditions. They widen during volatile market periods and contract during less volatile periods. Bollinger Bands become moving standard deviation bands. Bollinger Bands are displayed with a third line. This is the simple (short-term) moving average line. The time period for this moving average can vary. The default for short-term moving average in AmiBroker is 15 days.

An important thing to keep in mind is that Bollinger Bands do not generate buy and sell signals alone. They should be used with another indicator. RSI, for example, is quite good choice as a companion for Bollinger bands. When price touches one of the bands, it could indicate one of two things. It could indicate a continuation of the trend; or it could indicate a reaction the other way. So Bollinger Bands used by themselves do not provide all of what technicians need to know. Then RSI, which is an excellent indicator with respect to overbought and oversold conditions, comes with help. Generally, when price touches the upper Bollinger Band, and RSI is below 70, we have an indication that the trend will continue. Conversely, when price touches the lower Bollinger Band, and RSI is above 30, we have an indication that the trend should continue. If we run into a situation where price touches the upper Bollinger Band and RSI is above 70 (possibly approaching 80) we have an indication that the trend may reverse itself and move downward. On the other hand, if price touches the lower Bollinger Band and RSI is below 30 (possibly approaching 20) we have an indication that the trend may reverse itself and move upward. Avoid the trap of using several different indicators all working off the same input data. If you're using RSI with the Bollinger Bands, don't use MACD too. They both rely on the same inputs. You might consider using On Balance Volume, or Money Flow. RSI, On Balance Volume, and Money Flow, rely on different inputs.

## Indicators

### What is an indicator?

An indicator is a mathematical calculation that can be applied to a security's price and/or volume fields. The result is a value that is used to anticipate future changes in prices.

AmiBroker has following indicators built-in:

- [ROC](#)
- [RSI](#)
- [MACD](#)
- [CCI](#)
- [OBV](#)
- [NVI](#)
- [MFI](#)
- [Accumulation/Distribution](#)
- [TRIX](#)
- [Chaikin](#)
- [Relative Strength](#)
- [Ultimate Oscillator](#)
- [Stochastic](#)
- [TRIN \(Arms Index\)](#)
- [AD-Line \(Advance/Decline line\)](#)
- [Volume At Price histogram \(Volume Profile\)](#)
- [Relative Performance](#)

### Accumulation/Distribution

Accumulation/Distribution is a momentum indicator which takes into account changes in price and volume together. The idea is that a change in price coupled with an increase in volume may help to confirm market momentum in the direction of the price move.

Note the similarity of this formula to that of the stochastic; this is basically a stochastic multiplied by volume. This means that if the security closes to its high, the volume multiplier will be greater than if the security closes



nearer to its low.

If the Accumulation/Distribution indicator is moving up the buyers are driving the price move and the security is being accumulated. A decreasing A/D value implies that the sellers are driving the market and the security is being distributed. If divergence occurs between the Accumulation/Distribution indicator and the price of the security a change in price direction is probable.

The Accumulation/Distribution Line formula is as follows:

$$\frac{(CLOSE - LOW) - (HIGH - CLOSE)}{HIGH - LOW} \times VOLUME + I$$

Where *I* is yesterday's Accumulation/Distribution value.

## Advance–Decline line (AD–Line)

Line measuring advances and declines that reflects market breadth. In its simplest form ADLine is a summation over time of the net daily difference between the number of advancing issues and the number of declining issues. AmiBroker uses slightly improved formula which takes into account also number of unchanged issues. The exact AFL formula for AmiBroker's ADLine is:

```
Difference = ( AdvIssues() - DecIssues() ) / ( UncIssues() + 1 );
DiffSqrt = IIF( Difference > 0, sqrt( Difference ), - sqrt( - Difference ) );
ADLine = Cum( DiffSqrt );
```

This is a classical indicator which tends to give a good reading of the overall strength of the market. A break in the A/D line usually proceeds a break in prices. Look for non–confirmation and divergence.

See also AFL Function reference: [AFL Function: adline\(\)](#)

## ADX / Directional Movement Index

The ADX Indicator, otherwise known as Directional Movement Index.

The ADX is a trend following system. The average directional movement index, or ADX, determines the market trend. When used with the up and down directional indicator values, +DI and –DI, the ADX is an exact trading system. The standard interpretation for using the ADX (blue line) is to establish a long position whenever the +DI (green line) crosses above the –DI (red line). You reverse that position, liquidate the long position and establish a short position, when the –DI crosses above the +DI. In addition to the crossover rules, you must also follow the extreme point rule. When a crossover occurs, use the extreme price as the reverse point. For a short position, use the high made during the trading interval of the crossover. Conversely, reverse a long position using the low made during the trading interval of the crossover. You maintain the reverse point, the high or low, as your market entry or exit price even if the +DI and the –DI remain crossed for several

trading intervals. This is supposed to keep you from getting whipsawed in the market. For some traders, the most significant use of the ADX is the turning point concept. First, the ADX must be above both DI lines. When the ADX turns lower, the market often reverses the current trend. The ADX serves as a warning for a market about to change direction. The main exception to this rule is a strong bull market during a blow-off stage. The ADX turns lower only to turn higher a few days later. According to the developer of the DMI, you should stop using any trend following system when the ADX is below both DI lines. The market is in a choppy sideways range with no discernible trend. If you need further explanation, please refer to the author's original work. The book titled *New Concepts in Technical Trading Systems* by J. Welles Wilder, Jr. explains this indicator and several others.

## CCI – Commodity Channel Index

A price momentum indicator developed by Donald R. Lambert – it measures price excursions from the mean price as a statistical variation. The indicator works quite well with commodities, stocks and mutual funds. It keeps trades neutral in a sideways moving market, and helps get in the market when a breakout occurs.

A description of the CCI formula is as follows:

First, Calculate each periods mean. This is the high, plus the low, plus the close, divided by 3.

Second, calculate the n period simple moving average of these means.

Third, from each periods mean price, subtract the n period simple moving average of mean prices.

Fourth, Compute the mean deviation. This is the differences between each period's mean price and the n period simple moving average of those mean prices.

Fifth, Multiply the mean deviation by .015.

Sixth, the mean price, which we calculated in step three, is divided by .015 times the mean deviations from step 5.

Ordinarily, CCI ranges in value from +100 to –100. The rules are to buy and go long when CCI crosses above +100 and close the long when CCI falls back below +100. Conversely, sell short when CCI crosses below –100 and close the short when CCI crosses back above –100.

## Chaikin Oscillator

Developed by Marc Chaikin back in the early 1970's when opening prices were eliminated from many newspaper listings making it more difficult to calculate William's OBV. Chaikin substituted the average price  $[(HIGH+LOW)/2]$  for William's opening price and created an oscillator using 10–period and 3–period exponential moving averages of the resulting Accumulation/Distribution Line.

The basic premise of the Accumulation/Distribution Line is that the degree of buying or selling pressure can be determined by the location of the close, relative to the high and low for the corresponding period. There is buying pressure when a stock closes in the upper half of a period's range and there is selling pressure when a stock closes in the lower half of the period's trading range.

### Bullish Signals

There are two bullish signals that can be generated from the Chaikin Oscillator: positive divergences and

centerline crossovers. Because the Chaikin Oscillator is an indicator of an indicator, it is prudent to look for confirmation of a positive divergence, by a bullish moving average crossover for example, before counting this as a bullish signal.

### Bearish Signals

In direct contrast to the bullish signals, there are two bearish signals that can be generated from the Chaikin Oscillator: a negative divergence and a bearish centerline crossover. Allow a negative divergence to be confirmed by a bearish centerline crossover, before a bullish signal is rendered.

The Chaikin Oscillator is good for adding momentum to the Accumulation/Distribution Line, but can sometimes add a little too much momentum and be difficult to interpret. The moving averages are both relatively short and will therefore be more sensitive to changes in the Accumulation/Distribution Line. Sensitivity is important, but one must also be able to interpret the indicator.

## MACD – Moving Average Convergence/Divergence

This indicator uses three exponential moving averages, a short or fast average, a long or slow average and an exponential average of their difference, the last being used as a signal or trigger line. To fully understand the basics of MACD you must first understand simple moving averages. The Moving Average Convergence/Divergence indicator measures the intensity of public sentiment and is considered by Gerald Appel, its developer, to be a very good indicator signaling market entry points after a sharp decline. This indicator reveals overbought and oversold conditions and generates signals that predict trend or price reversals. It provides a sensitive measurement of the intensity of public sentiment and can be applied to the stock market, to individual stocks or to mutual funds. In some instances, it can provide advance warning of reversals allowing you to buy into weakness and sell into strength.

The Moving Average Convergence/Divergence indicator (MACD) is calculated by subtracting the value of 26-day exponential moving average from a 12-day exponential moving average. A 9-day exponential moving average (the "signal line") is automatically displayed on top of the MACD indicator line.

The basic MACD trading rule is to sell when the MACD falls below its 9-day signal line. Similarly, a buy signal occurs when the MACD rises above its signal line.

## Money Flow Index

The Money Flow Index (MFI) attempts to measure the strength of money flowing in and out of a security. It is closely related to the Relative Strength Index (RSI). The difference between the RSI and Money Flow is that where RSI only looks at prices, the Money Flow Index also takes volume into account.

Calculating Money Flow is a bit more difficult than the RSI.

First we need the average price for the day then we need the Money Flow:

$$\text{MoneyFlow} = \text{Volume} \times \text{Average Price}$$

Now, to calculate the money flow ratio you need to separate the money flows for a period into positive and negative. If the price was up in a particular day this is considered to be "Positive Money Flow". If the price closed down it is considered to be "Negative Money Flow".

$$\text{MoneyRatio} = \frac{\text{Positive MoneyFlow}}{\text{Negative MoneyFlow}}$$

It is the Money Flow Ratio which is used to calculate the Money Flow Index.

$$MFI = 100 - \frac{100}{100 + \text{MoneyRatio}}$$

The Money Flow ranges from 0 to 100. Just like the RSI, a stock is considered overbought in the 70– 80 range and oversold in the 20–30 range.

The shorter number of days you use, the more volatile the Money Flow is. The default is to use a 14 day average.

The interpretation of the Money Flow Index is as follows:

- Look for divergence/failure swings between the indicator and the price action. If the price trends higher (lower) and the MFI trends lower (higher), then a reversal may be imminent.
- Look for market tops to occur when the MFI is above a specific level (e.g., 80). Look for market bottoms to occur when the MFI is below a specific level (e.g., 20).

## Negative Volume Index

This indicator makes a very important assumption. It assumes that the unsophisticated investor follows market trends thus pushing up volume as they jump in on a rising security price. On the other hand, informed buying and selling by those "in the know" occurs on quieter periods reflected by negative volume changes on days of declining volume. This is an excellent bull market trend predictor. This index simply measures the trend of prices during periods when the volume is declining.

The price index is only adjusted on those days during which the volume has decreased from the previous day. If the volume did not change or was positive, the indicator remains unchanged. If the index rises, it means simply that the price of the security has gone up on a day that the volume has dropped. A drop in the index indicates that the price of the security has gone down while the volume declined. (The change in the index is calculated as a percentage change in the price).

This indicator can be compared to its longer period averages to reflect the movement of smart money. If, for example current index readings are above a six-month average, it can very well indicate an up trend for the market or the security.

## OBV – On Balance Volume

OBV was created by Joe Granville, the father of OBV analysis. This is a running total of volume that relates price changes and volume and shows accumulation and distribution action.

The classic OBV is calculated by adding today's total volume to a cumulative total when price closes higher than yesterday's close and subtracting today's total volume from the cumulative when the price closes lower than yesterday's close. If price remains the same, then the OBV is not changed. The actual amount of the price change is irrelevant and only the direction of change is significant for these calculations.

This indicator defines trends by showing underlying strength of price movements over time. A solid price trend is assumed to be accompanied with a stronger volume movement in the same direction. OBV analysis assumes that volume trends lead price trends and that OBV changes generally precede price changes. Look for divergence or non-confirmation between price and volume movements. A stock that is trending in an upward direction and starts to experience higher volume on days of lower closing prices usually indicates an

end to the current trend. Look for changes or breakouts in OBV trends. Sell short when the OBV makes a downside breakout and buy long when the on OBV upside breakouts.

## Parabolic SAR (Stop–And–Reverse)

Developed by Welles Wilder, creator of RSI and DMI, the Parabolic SAR sets trailing price stops for long or short positions. Also referred to as the stop–and–reversal indicator (SAR stands for "stop and reversal"), Parabolic SAR is more popular for setting stops than for establishing direction or trend. Wilder recommended establishing the trend first, and then trading with Parabolic SAR in the direction of the trend. If the trend is up, buy when the indicator moves below the price. If the trend is down, sell when the indicator moves above the price.

The formula is quite complex, but interpretation is relatively straightforward. The dotted lines below the price establish the trailing stop for a long position and the lines above establish the trailing stop for a short position. At the beginning of the move, the Parabolic SAR will provide a greater cushion between the price and the trailing stop. As the move gets underway, the distance between the price and the indicator will shrink, thus making for a tighter stop–loss as the price moves in a favorable direction.

There are two variables: the step and the maximum step. The higher the step is set, the more sensitive the indicator will be to price changes. If the step is set too high, the indicator will fluctuate above and below the price too often, making interpretation difficult. The maximum step controls the adjustment of the SAR as the price moves. The lower the maximum step is set, the further the trailing stop will be from the price. Wilder recommends setting the step at .02 and the maximum step at .20.

## RS – Relative Strength (comparative)

Compares the performance trend of a stock or industry group relative to another stock, group or index. This comparison removes the emotion from the marketplace. Many times a drop in relative strength can indicate a coming drop in actual price of the security. Do not confuse with Wilders's RSI.

The concept is to identify which stock or market sector is performing the best. Assuming that trends will continue to persist for some time, it is more probable that before a stock price will drop sharply it will first loose relative strength against other stocks. This would indicate a sell prior to such a price drop. An increase in relative strength does not necessarily indicate that the index is heading up, but it does signal a buy alert.

## RSI – Relative Strength Index

A technical indicator developed by Welles Wilder to help investors gauge the current strength of a security price relative to its past performance. The RSI is an excellent overbought/oversold indicator that can be used to predict trend reversal points. Do not confuse this index with relative strength in its everyday definition as used in comparing the movement of one security, index or group against the movement of another security, index or group. Developed by J. Welles Wilder, Jr. and first described in his book "New Concepts in Technical Trading Systems", this is a momentum oscillator that measures the velocity of directional price movement.

It compares a security highest highs and lowest lows over a period of time. RSI is based upon the difference between the average of the closing price on up days vs. the average closing price on the down days.

$$RSI = 100 - [100 / (1 + U/D)]$$

U = average of upward price closes (EMA of gains)

D = average of downward price closes (EMA of losses)

The ratio between up and down closing averages is in fact the makeup of the index. The time period specified determines the volatility of the RSI. For example, a 9-day time period will be more volatile than a 21-day time span. The author (Wilder) uses an n value of 14 days but other values may be used that better fit particular securities. The 9-day and 25-day RSIs have also gained popularity. Because you can vary the number of time periods in the RSI calculation, I suggest that you experiment to find the period that works best for you.

The RSI is a price-following oscillator that ranges between 0 and 100. A popular method of analyzing the RSI is to look for a divergence in which the market index is making a new high, but the RSI is failing to surpass its previous high. This divergence would be an indication of an impending reversal. When the RSI then turns down and falls below its most recent trough, it is said to have completed a failure swing. The failure swing would be considered a confirmation of an impending reversal.

In Mr. Wilder's book, he discusses five uses of the RSI in analyzing commodity charts (these apply to indices as well):

1. **Tops and Bottoms:** RSI readings above 70 indicate the shares are overbought and are likely to start falling. Readings below 30 indicate the shares are oversold and a rally can be expected. (AmiBroker automatically draws horizontal lines at these levels). The RSI usually forms these tops and bottoms before the underlying price chart.
2. **Chart Formations:** The RSI often forms chart patterns (such as head and shoulders or rising wedges) that may or may not be visible on the price chart.
3. **Failure Swings:** This is where the RSI surpasses a previous high (peak) or falls below a recent low (trough).
4. **Support and Resistance:** The RSI shows, sometimes more clearly than the price chart, levels of support and resistance.
5. **Divergence:** As discussed above, this occurs when the price makes a new high (or low) that is not confirmed by a new RSI high (or low).

## **ROC – Price Rate Of Change**

This indicator displays the rate-of-change of a security's price. Change is displayed as a percentage rather than as a ratio.

ROC is calculated by dividing the price change over the last n-periods by the closing price n-periods ago. This gives you percentage that the price has changed in the last n-periods.

When the 10-day ROC line is above the central line, the price is higher today than it was 10 periods ago. When the ROC line is below the central line, the price is lower today than it was 10 days ago. If the ROC line is above the central line, the price is higher than it was 10 days ago. If the ROC line is below the central line but rising, the price is still lower today than it was 10 days ago, but the range is narrowing.

The 12-day ROC is best used as a short to intermediate-term overbought/oversold indicator. The higher the ROC, the more overbought the security; the lower the ROC, the more likely a rally. However, as with all overbought/oversold indicators, it is best to wait for the market to begin to correct (i.e., turn up or down) before placing your trade. A market that appears overbought may remain overbought for some time. In fact, extremely overbought/oversold readings usually imply a continuation of the current trend.

The 12-day ROC tends to be very cyclical, oscillating back and forth in a fairly regular pattern. Often, price

changes can be anticipated by studying the previous cycles of the ROC and relating the previous cycles to the current market.

The optimum overbought/oversold levels (e.g., +/-5) will vary depending on the security being analyzed and overall market conditions. In strong bull markets, it is usually beneficial to use higher levels, perhaps +10 and -5.

## Stochastic Slow

Stochastic is an oscillator that measures the position of a stock or security compared with its recent trading range indicating overbought or oversold conditions.

It displays current day price at a percentage relative to the security's trading range (high/low) over the specified period of time.

$$FastStoc = \% K = \frac{(\text{today's close}) - (\text{low price in period } n)}{(\text{high price in period } n) - (\text{low price in period } n)}$$

In a Slow Stochastic, the highs and lows are averaged over a slowing period. The default is usually 3 for slow and 1 (no slowing) for fast. The line can then be smoothed using an exponential moving average, Weighted, or simple moving average %D. Confirming Buy/sell signals can be read at intersections of the %D with the %K as well.

The Stochastic Oscillator always ranges between 0% and 100%. A reading of 0% shows that the security's close was the lowest price that the security has traded during the preceding x-time periods. A reading of 100% shows that the security's close was the highest price that the security has traded during the preceding x-time periods. When the closing price is near the top of the recent trading range (above 80%), the security is in an overbought condition and may signal for a possible correction. Oversold condition exists at a point below %20. Prices close near the top of the range during uptrends and near the bottom of the range during downtrends.

## TRIN – Arms Index

Trading Index, a technical measure of advances and declines within the market. TRIN takes into account the number and volume of issues that advanced in price, and the number and volume of issues that declined in price. This index measures the relative strength of volume associated with advancing stocks against the strength of volume associated with declining stocks.

Exact AFL formula for TRIN is:

```
ArmsIndex = ( AdvIssues() / DecIssues() ) / ( AdvVolume() / DecVolume ) ;
```

A TRIN value of 1 indicates that the ratio of up volume to down volume is equal to the ratio of advancing issues to the declining issues and the market is in a neutral condition. A neutral condition simply means that the up volume is equally distributed over the advancing issues and that the down volume is equally distributed over declining issues for the day.

This indicator, although simple in its formulation, requires much study in its application. There are many variations applied to the TRIN. Many analysts use a 10-day moving average of TRIN as an indicator. AmiBroker plots two different averages for TRIN with the default averaging periods of 15 and 45. A reading of less than 1.0 usually indicates a bullish demand while a reading greater than 1 can signify a bearish market

condition. It must be kept in mind that the indicator behavior and its reading and interpretation depends on whether the market is in a bullish or bearish phase. The actual time duration of this market phase must also be considered. Do not attempt to make and buy or sell decisions based on movements of this indicator by itself.

See also AFL Function reference: [AFL Function: trin\(\)](#)

## TRIX – TRIPLE eXponential

TRI-ple eXponential. TRIX displays the % rate-of-change of a triple exponentially smoothed moving average of the closing price of a security.

TRIX is calculated as a one period rate of change of the third exponential moving average pass of the closing price.

TRIX is designed to filter out insignificant cycles – those smaller than the number of moving averages specified. The TRIX indicator oscillates around a zero line. Trades should be placed when the indicator changes direction.

## Ultimate Oscillator

Larry Williams, the designer of the Ultimate Oscillator, wanted to address the problems experienced with most oscillators when used over different lengths of time.

Ultimate oscillator signals are the following: divergence and a breakout in the Oscillator's trend, as well as overbought and oversold levels.

The value of other oscillators can vary greatly depending on the number of time periods used during the calculation. So, the Ultimate Oscillator, uses weighted sums of three oscillators which represent short, intermediate, and long term market cycles (7, 14, & 28-period), and it is plotted as a single line on a vertical scale of 0 to 100.

The three components are based on Williams's definitions of buying and selling "pressure."

A trade should be initiated following a divergence and a breakout in the Ultimate Oscillator's trend.

### Signals:

*A Buy signal is generated when:*

A positive or bullish divergence occurs between the Ultimate oscillator and the price.

The Ultimate falls below 30 and then rises above the previous high established during the divergence (the actual buy signal).

*A Sell Signal is offered when:*

A negative or bearish divergence occurs between the Ultimate and the price.

The Ultimate rises above 70 and then falls below the previous low established during the divergence (the actual sell signal).

*Closing existing positions:*

Close long positions when the Ultimate exceeds 70.

Close short positions when the Ultimate goes below 30.

As with most indicators, it is good if these signals are confirmed by other indicators before being acted upon.



## VAP – Volume At Price histogram

Volume At Price histogram is also known as "Volume Profile" chart.

To turn it on simply go to **Tools→Preferences** and change Type of the VAP from "NONE" to "Left-side solid area chart, behind" for example

VAP shows total volume of trading that occurred at given price level. VAP is calculated from data bars that are currently visible.

Actual algorithm involves not ONE price but High–Low price RANGE.

AmiBroker DISTRIBUTES equally bar's volume over High–Low range to produce VAP histogram. For example if bar's volume is 10000 and H–L range spans 3 "lines" of VAP histogram than each of 3 lines involved gets added  $10000/3$  to produce statistics. This gives much more accurate results than using single price as some other implementations do.

To turn VAP on/off use: Tools→Preferences→Main chart

You can also add VAP to your own custom charts using [PlotVAPOverlay](#) AFL function.

## Relative Performance chart

Relative Performance chart compares the rate of price change of two or more tradable instruments. Plot starts with 0% at the very first visible bar and shows percentage change of closing price since that point for every symbol in the list. Relative performance charts are great for comparing dissimilarly priced issues (for example stocks and indices) since it displays percentage changes, not absolute values. You can easily see which instruments perform better than others and choose best performers for your trading.

You can adjust the list of symbols that are plotted in the Relative Performance chart by clicking with RIGHT mouse button over the chart and choosing "Parameters" item from the context menu. In the Parameters dialog you can enter a comma-separated list of symbols that you want to get the chart for. There is no limit on number of symbols you can enter, but please remember to separate symbols by comma and not using spaces unless symbol itself has them.

# AmiBroker Formula Language (AFL)

AmiBroker is equipped with a powerful formula language allowing you to write trading system rules, define your own indicators and custom commentaries. This chapter explains the language, gives you detailed reference of built-in analysis functions and shows how to use AFL-tools such as automatic analyzer and formula editor .

- Language Reference
  - ◆ [Basics \(lexical elements, predefined variables\)](#)
  - ◆ [Keywords](#)
- Function Reference
  - ◆ [Alphabetical list of all AFL functions](#)
  - ◆ [Categorized list of AFL functions](#)
  - ◆ [AddToComposite function](#) – creating multiple security statistics
  - ◆ [Equity function](#) – analysing your trading system performance
  - ◆ [Variable-period functions](#)
- [User-defined functions and variable scope](#)
- [AFL Tools](#)
- [AFL Scripting Host](#)
- [Component Object Model support in AFL](#)
- [Common coding mistakes](#)
- [Advanced portfolio backtester interface](#)
- [Adding custom backtester metrics](#)
- [Using Low-level graphics functions](#)

See also: [Tutorial: Understanding how AFL works](#)

# AFL Reference Manual

Revision 2.90

## Introduction

AFL is a special programming language used to define and create custom indicators, scans, explorations, back-tests and guru commentaries.

## Basics

### Lexical elements

This chapter describes the different categories of word-like units (tokens) recognized by the AFL language interpreter.

#### Whitespace

*Whitespace* is the collective name given to spaces (blanks), tabs, new line characters and comments. Whitespace can serve to indicate where tokens start and end, but beyond this function, any surplus whitespace is discarded.

#### Comments

*Comments* are pieces of text used to annotate a program. Comments are for the programmer's use only; they are stripped from the source code before parsing. There are two ways to delineate comments: C-like comments and C++ like comments. A C-like comment is any sequence of characters placed after the symbol pair `/*`. The comment terminates at the first occurrence of the pair `*/` following the initial `/*`. The entire sequence, including the four comment-delimiter symbols, is replaced by one space. A C++ like comments are single-line comments that start by using two adjacent slashes (`//`) in any position within the line and extend until the next new line.

AFL does not allow nested comments.

#### Tokens

AFL recognizes five classes of tokens:

- identifiers
- constants
- string-literals
- operators
- punctuators (also known as separators)

*Identifiers* are arbitrary names of any length given to functions and variables. Identifiers can contain the letters (a–z, A–Z), the underscore character ("`_`"), and the digits (0–9). The first character must be a letter. AFL identifiers are NOT case sensitive.

*Constants* are tokens representing fixed numeric or character values. Numeric constants consist of decimal integer and optionally: decimal point and decimal fraction part. Negative numeric constants have unary minus

(-) prefixed.

String constants, also known as *string literals*, form a special category of constants used to handle fixed sequences of characters and are written as a sequence of any number of characters surrounded by double quotes:

```
"This is literally a string"
```

The null (empty) string is written "". The characters inside the double quotes can include escape sequences ("\\n" – a new line escape sequence).

A *Constant expression* is an expression that always evaluates to a constant. They are evaluated just as regular expressions are.

*Punctuator* (also known as separator) in AFL is one of the following characters:

```
( ) , ; = .
```

*Parentheses* (open ( and close ) ) group expressions, isolate conditional expressions and indicate function calls and function parameters:

```
d = c * ( a + b ) /* override normal precedence */
a = ( b AND c ) OR ( d AND e ) /* conditional expression */
func() /* function call no arguments */
```

The *comma* (,) separates the elements of a function argument list

The *semicolon* (;) is a statement terminator. Any legal AFL expression followed by a semicolon is interpreted as a statement, known as expression statement. The expression is evaluated and its value is discarded (except Guru Commentaries where string values are written to output window)

The *dot* (.) is a member access operator. It is used to call COM object methods. If myobj variable holds the object, using dot operator we can call the methods (functions) of myobj object:

```
myobj.Method();
```

The *equal sign* (=) separates variable declarations from initialization lists:

```
x = 5;
```

It also indicates the default value for a parameter (see built-in function description):

```
macd( fast = 12; slow = 26 ) /* default values for fast and slow arguments
```

## Language structure

Each formula in AFL contains of one or more expression statements. Each statement **MUST** be terminated by semicolon (;). In this way you are able to break long expressions into several physical lines (in order to gain clarity) and AmiBroker will still treat it like a single statement until terminating semicolon. Examples:

```
x = ( y + 3 );          /* x is assigned the value of y + 3 */
x = y = 0;             /* Both x and y are initialized to 0 */
proc( arg1, arg2 );    /* Function call, return value discarded */
y = z = ( f( x ) + 3 ); /* A function-call expression */
```

```
my_indicator =      IIf( MACD() > 0,
                      Close - MA(Close,9),
                      MA( Close, 9 ) - Close );
/* one statement in several lines */
```

## Identifiers

Identifiers in AFL are used to identify variables and functions.

There are some predefined identifiers referencing built-in arrays and functions.

The most important are *price array identifiers*. They identify specific price fields that the formula should operate on. The valid price array identifiers are **open**, **high**, **low**, **close**, **volume**, **openint**, **average**. Price array identifiers can be abbreviated as shown in the following table. Note that these are not case-specific.

Long name	Abbreviation	Comment
Open	O	
High	H	
Low	L	
Close	C	
Volume	V	
OpenInt	OI	
Avg	<none available>	(High+Low+Close)/3 – so called "typical price"

Examples of the use of price array identifiers in formulas are shown below.

```
MA( Close, 10 ); IIf( H > Ref(H,-1), MA(H,20), MA(C,20) );
```

## Operators

### Comparison operators

Comparison operators are divided into two types:

- relational ( <, >, <=, >= )
- equality ( ==, != )

Symbol	Meaning
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
=	Equal to
!=	Not equal to

These operators give true (1) or false (0) value as a result of comparison.

## Assignment operator

Symbol	Meaning
=	Store the value of the second operand in the object specified by the first operand ( simple assignment ).

The assignment operator assigns a value to a variable:

```
result = expression;
```

where *result* is variable identifier and *expression* is any numerical, array or text expression.

As the = operator behaves like other operators, expressions using it have a value in addition to assigning that value into variable. This means that you can chain assignment operators as follows:

```
j = k = l = 0;
```

j, k, and l equal zero after the example statement is executed.

Attention: please DO NOT confuse assignment operator (=) with equality check (==)

These are two different operators and you must not use assignment (=) to check for equality.

```
if( Name() = "MSFT" ) // WRONG !!! – variable assignment operator used instead of equality check
{
}
```

```
if( Name() == "MSFT" ) // CORRECT – equality operator used properly
{
}
```

This is one of common coding mistakes listed [here](#).

## Arithmetic operators

Formulas can contain the following mathematical operators:

Symbol	Meaning
+	Addition
–	Subtraction (or negative value)
*	Multiplication
/	Division
%	Modulus (or remainder) (AFL 1.7+)
^	Exponentiation (raising to a power)
	Bit-wise "Or" (AFL 2.1+)
&	Bit-wise "And" (AFL 2.1+)

The following formulas illustrate the use of operators in a formula:

```
var1 = ( H + L ) / 2;
```

```
var2 = MA(C,10)-MA(C,20) / (H + L + C);
```

```
var3 = Close + ((1.02 * High)-High);
```

#### Logical operators

Symbol	Meaning
<b>NOT</b>	Logical "Not" – gives "True" when operand is equal to false
<b>AND</b>	Logical "And" – gives "True" result if BOTH operands are true at the same time
<b>OR</b>	Logical "Or" – gives "True" result if ANY of operands is true

If a formula requires multiple conditions, you can combine the conditions with AND and OR operators. For example, maybe you'd like to plot a +1 when the MACD is greater than zero and the RSI is greater than 70:

```
Condition = MACD() > 0 AND RSI(14) > 70;
```

You can add as many conditions within a formula as you like.

#### Compound assignment operators

Introduced in version 5.00, the compound operators are specified in the form of:

```
destinvar op= expr;
```

where *destinvar* is the variable, *expr* is the expression, and *op* is one of the following arithmetic operators: +, -, \*, /, %, &, |

The *destinvar op= expr* form behaves as:

```
destinvar = destinvar op expr;
```

This is shortcut form for common assignment statements like *k = k + 2*; so you can write it shorter as:

```
k += 2;
```

and it will work the same but little faster.

Full list of available assignment operators is here:

No	Symbol	Meaning
1	=	Store the value of the second operand in the object specified by the first operand ( simple assignment ).
2	*=	Multiply the value of the first operand by the value of the second operand; store the result in the object specified by the first operand.
3	/=	Divide the value of the first operand by the value of the second operand; store the result in the object specified by the first operand.

4	<b>%=</b>	Take modulus of the first operand specified by the value of the second operand; store the result in the object specified by the first operand.
5	<b>+=</b>	Add the value of the second operand to the value of the first operand; store the result in the object specified by the first operand.
6	<b>=</b>	Subtract the value of the second operand from the value of the first operand; store the result in the object specified by the first operand.
7	<b>&amp;=</b>	Obtain the bitwise AND of the first and second operands; store the result in the object specified by the first operand.
8	<b> =</b>	Obtain the bitwise inclusive OR of the first and second operands; store the result in the object specified by the first operand

### Operator precedence and the parentheses

AFL supports parentheses in formulas.

Parentheses can be used to control the operation precedence (the order in which the operators are calculated). AmiBroker always does operations within the innermost parentheses first. When parentheses are not used, the precedence is as follows (higher precedence listed first):

No	Symbol	Meaning
1	<b>++</b>	Post-increment/pre-increment (i++ works like i = i + 1)
2	<b>--</b>	Post-decrement/pre-decrement (i-- works like i = i - 1)
3	<b>[ ]</b>	Array element (subscript) operator
4	<b>^</b>	Exponentiation
5	<b>-</b>	Negation – Unary minus
6	<b>*</b>	Multiplication
7	<b>/</b>	Division
8	<b>%</b>	Reminder (Modulo operator)
9	<b>+</b>	Addition
10	<b>-</b>	Subtraction
11	<b>&lt;</b>	Less than
12	<b>&gt;</b>	Greater than
13	<b>&lt;=</b>	Less than or equal to
14	<b>&gt;=</b>	Greater than or equal to
15	<b>==</b>	Equal to
16	<b>!=</b>	Not equal to



17	<b>&amp;</b>	Bit-wise "And" (AFL 2.1+)
18	<b> </b>	Bit-wise "Or" (AFL 2.1+)
19	<b>NOT</b>	Logical "Not"
20	<b>AND</b>	Logical "And"
21	<b>OR</b>	Logical "Or"
22	<b>=</b>	Variable assignment operator
23	<b>*= /= %= += -= &amp;=  =</b>	Compound assignment

The expression

```
H + L / 2;
```

(without parenthesis) would be calculated by AmiBroker as "L / 2" plus "H", since division has a higher precedence. This would result in a much different value than

```
(H + L) / 2;
```

A few words about increment/decrement operators. There are two kinds of them: postfix and prefix.

The unary operators (++ and --) are called prefix increment or decrement operators when the increment or decrement operators appear before the operand. Postfix increment and decrement has higher precedence than prefix increment and decrement operators. When the operator appears before its operand, the operand is incremented or decremented and its new value is the result of the expression.

```
i = 5;
```

```
j = ++i; // i will be incremented first and result (number 6) will be assigned to j.
```

The result of the postfix increment or decrement operation is the value of the postfix-expression before the increment or decrement operator is applied. The type of the result is the same as that of the postfix-expression but is no longer an l-value. After the result is obtained, the value of the operand is incremented (or decremented).

```
i = 5;
```

```
j = i++; // j will be assigned the value of 5 (before incrementation) and then i will be incremented to 6.
```

**Accessing array elements: [ ] – subscript operator**

An array identifier followed by an expression in square brackets ([ ]) is a subscripted representation of an element of an array object.

```
arrayidentifier [ expression ]
```

It represents the value of expression–th element of array.

**BarCount** constant gives the number of bars in array (such as Close, High, Low, Open, Volume, etc). Array elements are numbered from 0 (zero) to BarCount–1.

To get the first bar you can use array[ 0 ], to get the last bar of array you can use array[ BarCount – 1 ];

For example:

```
Close[ 5 ];
```

Represents the sixth element (bar) of the close array.

```
Close[ 0 ];
```

Represents the very first available bar of the close array.

```
High[ BarCount - 1 ];
```

Represents the last bar of High array.

**Compound statements (Blocks)**

A compound statement consists of zero or more statements enclosed in curly braces ({ }). A compound statement can be used anywhere a statement is expected. Compound statements are commonly called blocks.

```
{
    statement1;
    ....
    statementN;
}
```

(this is 'borrowed' from C language, users of other programming languages are used to use BEGIN for { and END for })

```
if( Amount > 100 )
{
    _TRACE("Amount above 100");
    Balance = Balance + Amount;
}
else
    Balance = Balance - Amount;
```

**Built-in Functions**

In addition to mathematical operators, AmiBroker contains over 70 built-in functions that perform mathematical operations.

The following formula consists of a single function that gives the square roots of the closing prices:

```
sqrt( Close );
```

The following formula consists of a single function that gives a 14-period RSI indicator:

```
Graph0 = RSI(14);
```

The following formula consists of two functions. The result is the difference between the MACD indicator and a 9-period exponential moving average of the MACD:

```
Graph0 = MACD() - EMA(MACD(),9);
```

All function calls must consist of function identifier (name) followed by a pair of parentheses.

As has been eluded to in earlier examples, a function can be "nested" within a function. The nested function can serve as the main function's data array parameter. The following examples show functions nested within functions:

```
MA( RSI(15), 10 );
```

```
MA( EMA( RSI(15), 20 ), 10 );
```

The first example calculates a 10-period simple moving average of a 15-period Relative Strength Index (RSI). The second example calculates a 20-period exponential moving average of a 15-period RSI, and then calculates a 10-period simple moving average of this moving average.

**Conditional function IIF()**

The iif() function is used to create **conditional assignments**. It contains three parameters as shown in the following example.

```
dynamicrsi = IIf( Close > MA(C,10), RSI(9), RSI(14) );
```

The above "iif" statement reads (in English) as follows: If today's close is greater than today's 10-day simple moving average of the close, then assign a 9-day RSI to the *dynamicrsi* variable, otherwise, assign a 14-day RSI. The next formula assigns "positive volume" to *volresult* variable if the close is greater than the median price. Otherwise, "negative volume" is assigned.

```
volresult = IIf( Close > (High+Low)/2, Volume, -Volume );
```

If you simply want an expression to be evaluated as either true or false, it can be done without the use of the iif() function. The following formula will result in either a 1 (true) or a 0 (false):

```
result = RSI(14) > 70;
```

The same done with iif() gives the same results, but the formula is longer.

```
result = IIf(RSI(14) > 70, 1, 0 );
```

Please note that IIF is a function – so the result of evaluation is returned by that function and should be assigned to some variable.

IIf always evaluates both TRUE\_PART and FALSE\_PART, even though it returns only one of them. Because of this, you should watch for undesirable side effects. **IIF function is NOT a flow-control statement.** If you need flow control (conditional execution of some code parts) you should look for **if-else** conditional statement described later in this document.

The following example shows one **common error** made with IIF function:

```
IIf( condition, result = 7, result = 9 ); // THIS IS WRONG
```

Correct usage is:

```
result = IIf( condition, 7, 9 );
```

*/\* 7 or 9 is \*returned\* and assigned to result variable depending on condition \*/*

## Variables

In order to shorten, simplify, enhance, and make the maintenance of complex formulas easier, you may want to use variables. In fact using variables you can significantly improve formula calculation speed. So it is strongly recommended to use variables and there is **no limit** on number of variables you can define.

A variable is an identifier that is assigned to an expression or a constant. The number of variables used in a formula is not limited. Variables must be assigned before the variable is used in the formula. Variables cannot be assigned within a function call.

User-defined variable names (identifiers) cannot duplicate names already used by functions (e.g., ma, rsi, cci, iif, etc.) or predefined array identifiers (e.g., open, high, low, close, simple, o, c, l, h, s, a).

## Reserved variables

AmiBroker uses some reserved variable names in its formulas, for example in Auto-Analysis window you have to assign values to 2 variables named 'buy' or 'sell' to specify the conditions where "buy" and "sell" conditions occur. For example (system that buys when MACD rises above 0 line, and sells when MACD falls below 0 line)

```
Buy = Cross( MACD(), 0 );
Sell = Cross( 0, MACD() );
```

AmiBroker uses the following reserved variable names. Please note that variables marked as obsolete should NOT be used in new coding. They are left for backward compatibility only and new formulas should use modern functions like Plot() to plot indicators and AddColumn() to define exploration columns.

Variable	Usage	Applies to
buy	defines "buy" (enter long position) trading rule	Automatic Analysis, Commentary

sell	defines "sell" (close long position) trading rule	Automatic Analysis, Commentary
short	defines "short" (enter short position – short sell) trading rule	Automatic Analysis
cover	defines "cover" (close short position – buy to cover) trading rule	Automatic Analysis
buyprice	defines buying price array (this array is filled in with the default values according to the Automatic Analyser settings)	Automatic Analysis
sellprice	defines selling price array (this array is filled in with the default values according to the Automatic Analyser settings)	Automatic Analysis
shortprice	defines short selling price array (this array is filled in with the default values according to the Automatic Analyser settings)	Automatic Analysis
coverprice	defines buy to cover price array (this array is filled in with the default values according to the Automatic Analyser settings)	Automatic Analysis
title	defines title text (overrides any graph/name)	Indicators
tooltip	Allows you to define your own text for data tooltip  Example:  Tooltip = "This is my tool tip text showing close price: " + Close;	Indicators
graphxspace	defines percentage extra space added at the top and the bottom of the chart	Indicators
graphzorder	GraphZOrder variable allows to change the order of plotting indicator lines. When GraphZOrder is not defined or is zero (false) – old ordering (last to first) is used, when GraphZOrder is 1 (true) – reverse ordering is applied.	Indicators
exclude	If defined, a true (or 1) value of this variable excludes current symbol from scan/exploration/back test. They are also not considered in buy and hold calculations. Useful when you want to narrow your analysis to certain set of symbols.	Automatic Analysis
roundlotsize	defines round lot sizes used by backtester (see explanations below)	Automatic Analysis (new in 4.10)
ticksize	defines tick size used to align prices generated by <b><i>built-in stops</i></b> (see explanations below) (note: it does not affect entry/exit prices specified by buyprice/sellprice/shortprice/coverprice)	Automatic Analysis (new in 4.10)
pointvalue	allows to read and modify future contract point value (see <a href="#">backtesting futures</a> )	Automatic Analysis (new in 4.10)
margindeposit	allows to read and modify future contract margin (see <a href="#">backtesting futures</a> )	Automatic Analysis (new in 4.10)
positionsiz	Allows control dollar amount or percentage of portfolio that is invested into the trade (more information available in the " <a href="#">Tutorial: Backtesting your trading ideas</a> ")	Automatic Analysis (new in 3.9)

positionscore	Defines the score of the position. More details: " <a href="#">Tutorial: Portfolio Backtesting</a> ")	Automatic analysis
numcolumns	Exploration only: defines the number of your own columns (excluding predefined ticker and date columns) and assign the column value to the variable	Automatic Analysis
filter	<p>Exploration only: controls which symbols/quotes are accepted. If "true" (or 1) is assigned to that variable for given symbol/quote it will be displayed in the report.</p> <p>So, for example, the following formula will accept all symbols with closing prices greater than 50 :</p> <pre>filter = close &gt; 50;</pre>	Automatic Analysis
columnN (obsolete)	<p>Exploration only: defines Nth column value. Example:</p> <pre>column0 = Close;</pre>	Automatic Analysis
columnNformat (obsolete)	<p>Exploration only: allows you to define the formatting applied to numbers. By default all variables are displayed with 2 decimal digits, but you can change this by assigning a different value to this variable: 1.5 gives 5 decimal digits, 1.0 gives no decimal digits. So, in our example, typing:</p> <pre>column0format = 1.4;</pre> <p>will give closing prices displayed with 4 decimal digits. (Note for advanced users: the integer part of this number can be used to pad formatted number with spaces – 6.0 will give no decimal digits but a number space–padded upto 6 characters.)</p>	Automatic Analysis
columnNname (obsolete)	<p>Exploration only: allows you to define the header name. Assigning</p> <pre>column0name = "Close";</pre> <p>will change the name of the first custom column from the default "Column 0" to more appropriate "Close".</p>	Automatic Analysis
maxgraph (obsolete)	specifies maximum number of graphs to be drawn in custom indicator window (default=3)	Indicators
graphN (obsolete)	defines the formula for the graph number N (where N is a number 0,1,2,..., maxgraph–1)	Indicators
graphNname (obsolete)	defines the name of Nth graph line. This will appear in the title of the chart pane	Indicators
graphNcolor (obsolete)	<p>defines the color index of Nth graph line (color indexes are related to the current palette – see Preferences/Color)</p> <pre>colorCustom1 = 0 colorCustom2 = 1 colorCustom3 = 2 colorCustom4 = 3 colorCustom5 = 4</pre>	Indicators

colorCustom6 = 5
colorCustom7 = 6
colorCustom8 = 7
colorCustom9 = 8
colorCustom10 = 9
colorCustom11 = 10
colorCustom12 = 11
colorCustom13 = 12
colorCustom14 = 13
colorCustom15 = 14
colorCustom16 = 15
colorBlack = 16
colorBrown = 17
colorDarkOliveGreen = 18
colorDarkGreen = 19
colorDarkTeal = 20
colorDarkBlue = 21
colorIndigo = 22
colorDarkGrey = 23
colorDarkRed = 24
colorOrange = 25
colorDarkYellow = 26
colorGreen = 27
colorTeal = 28
colorBlue = 29
colorBlueGrey = 30
colorGrey40 = 31
colorRed = 32
colorLightOrange = 33
colorLime = 34
colorSeaGreen = 35
colorAqua = 35
colorLightBlue = 37
colorViolet = 38
colorGrey50 = 39
colorPink = 40
colorGold = 41
colorYellow = 42
colorBrightGreen = 43
colorTurquoise = 44
colorSkyblue = 45
colorPlum = 46
colorLightGrey = 47
colorRose = 48
colorTan = 49
colorLightYellow = 50

	colorPaleGreen = 51 colorPaleTurquoise = 52 colorPaleBlue = 53 colorLavender = 54 colorWhite = 55	
graphMbarcolor (obsolete)	defines the array that holds palette indexes for each bar drawn	Indicators
graphNstyle (obsolete)	<p>defines the style of Nth graph. Style is defined as a combination (sum) of one or more following flags:</p> <p>styleLine = 1 – normal (line) chart (default)  styleHistogram = 2 – histogram chart  styleThick = 4 – fat (thick)  styleDots = 8 – include dots  styleNoLine = 16 – no line  styleLog = 32 – semi-logarithmic scale  styleCandle = 64 – candlestick chart  styleBar = 128 – traditional bar chart  styleNoDraw = 256 – no draw (perform axis scaling only)  styleStaircase = 512 – staircase (square) chart  styleSwingDots = 1024 – middle dots for staircase chart  styleNoRescale = 2048 – no rescale  styleNoLabel = 4096 – no value label  stylePointAndFigure = 8192 – point and figure  (new in 4.20):  styleArea = 16384 – area chart (extra wide histogram)  styleOwnScale = 32768 – plot is using independent scaling  styleLeftAxisScale = 65536 – plot is using left axis scale (independent from right axis)</p> <p>Not all flag combinations make sense, for example (64+1) (candlestick + line) will result in candlestick chart (style=64)</p> <p>Note on candlestick/bar charts: these styles use indirectly O, H, L arrays in addition to graphN. So ordinary candlestick price chart formula is graph0=close; graph0style=64;.  But if you want to draw something else than close price you have to assign new values to predefined O,H,L arrays.</p>	Indicators
graphMbarcolor (obsolete)	defines the array of color indexes for the bars and candlesticks in Nth graph line (color indexes are related to the current palette – see Preferences/Color)	Indicators

**SEE ALSO:**

- [KEYWORDS](#)
- [USER-DEFINABLE PROCEDURES, LOCAL/GLOBAL SCOPE](#)



## Keywords

The following are keywords in AmiBroker Formula Language:

Loops:

- `do` (part of do-while statement)
- `while`
- `for`

Conditional execution / Flow control:

- `if` (part of if-else statement)
- `else` (part of if-else statement)
- `switch`
- `break` (part of the switch statement or for/while statements)
- `case` (part of the switch statement)
- `continue` (part of for/while statements)
- `default` (part of switch statement)

Functions:

- `function`
- `procedure`
- `return`
- `local` (variable scope)
- `global` (variable scope)

## break Keyword

The break keyword is a part of `switch` statement and an optional part of looping `for` , `do-while` and `while` statements.

The break keyword terminates the smallest enclosing do, for, switch, or while statement in which it appears.

**break;**

The break statement is used to exit an iteration or switch statement. It transfers control to the statement immediately following the iteration substatement or switch statement.

The break statement terminates only the most tightly enclosing loop or switch statement. In loops, break is used to terminate before the termination criteria evaluate to 0. In the switch statement, break is used to terminate sections of code normally before a case label. The following example illustrates the use of the break statement in a for loop:

```
i = 0;
while ( i < 10 )
{
    i++;
    // break at step 5
    if( i == 5 )
    {
```

```

        break;
    }
    printf( "Step " + i );
}

```

For an example of using the break statement within the body of a switch statement, see [The switch Statement](#).

## case Keyword

The case keyword is an integral part of [switch–case statement](#).

## continue Keyword

The continue keyword is an optional part of [for](#) , [do–while](#) and [while](#) statements.

It stops the current iteration of a loop, and starts a new iteration.

### continue;

You can use the continue statement only inside a while, do...while, or for loop. Executing the continue statement stops the current iteration of the loop and continues program flow with the beginning of the loop.

This has the following effects on the different types of loops:

while and do...while loops test their condition, and if true, execute the loop again. for loops execute their increment expression, and if the test expression is true, execute the loop again.

The following example illustrates the use of the continue statement:

```

i = 0;
while ( i < 10 )
{
    i++;
    // Skip 5
    if( i == 5 )
    {
        continue;
    }
    printf( "Step " + i );
}

```

## default Keyword

The default keyword is an integral part of [switch–case statement](#).

## do Keyword

The **do** keyword is a part of **do–while** statement.

## do–while Statement

The **do–while** statement lets you repeat a statement or compound statement until a specified expression becomes false.

## Syntax

**do** *statement* **while** ( *expression* ) ;

The *expression* in a **do-while** statement is evaluated after the body of the loop is executed. Therefore, the body of the loop is always executed at least once.

The *expression* must have numeric or boolean type. Execution proceeds as follows:

1. The statement body is executed.
2. Next, *expression* is evaluated. If *expression* is false, the **do-while** statement terminates and control passes to the next statement in the program. If *expression* is true (nonzero), the process is repeated, beginning with step 1.

This is an example of the **do-while** statement:

```
x=100;
do
{
    y = sin( x );
    x--;
} while ( x > 0 );
```

In this **do-while** statement, the two statements `y = sin( x );` and `x--;` are executed, regardless of the initial value of `x`. Then `x > 0` is evaluated. If `x` is greater than 0, the statement body is executed again and `x > 0` is reevaluated. The statement body is executed repeatedly as long as `x` remains greater than 0. Execution of the **do-while** statement terminates when `x` becomes 0 or negative. The body of the loop is executed at least once.

## else Keyword

The **else** keyword is an optional part of if-else statement.

## if, else Statement

```
if( expression )
statement1
[else
statement2]
```

The **if** keyword executes *statement1* if *expression* is true (nonzero); if **else** is present and *expression* is false (zero), it executes *statement2*. After executing *statement1* or *statement2*, control passes to the next statement.

### Example 1

```
if ( i > 0 )
    y = x / i;
else
{
    x = i;
    y = abs( x );
}
```

```
}
```

In this example, the statement  $y = x/i$ ; is executed if  $i$  is greater than 0. If  $i$  is less than or equal to 0,  $i$  is assigned to  $x$  and  $\text{abs}(x)$  is assigned to  $y$ . Note that the statement forming the **if** clause ends with a semicolon.

When nesting **if** statements and **else** clauses, use braces to group the statements and clauses into compound statements that clarify your intent. If no braces are present, the compiler resolves ambiguities by associating each **else** with the closest **if** that lacks an **else**.

### Example 2

```
if ( i > 0 )           /* Without braces */
    if ( j > i )
        x = j;
    else
        x = i;
```

The **else** clause is associated with the inner **if** statement in this example. If  $i$  is less than or equal to 0, no value is assigned to  $x$ .

### Example 3

```
if ( i > 0 )
{
    /* With braces */
    if ( j > i )
        x = j;
}
else
    x = i;
```

The braces surrounding the inner **if** statement in this example make the **else** clause part of the outer **if** statement. If  $i$  is less than or equal to 0,  $i$  is assigned to  $x$ .

### Common misunderstandings

*"New if-else problem"*

Question:

Why I get the syntax error when I write:  $\text{if}(H > \text{Ref}(H, -1))$

Answer:

if-else statement changes flow of execution (opposite to IIF function that evaluates all arguments and works on arrays) and you can not really write

```
if ( H > Ref(H, -1) )
```

because it has no meaning. It would translate to "If high array is higher than high array shifted one bar" (see tutorial below). Flow control statement (such as if-else) has to get SINGLE boolean value to make decision which execution path should be taken. If you write  $H$  (or High) it means ARRAY (entire array).

if you write  $H[i]$  – it means  $i$ -th element of the array. The subscript operator  $[ ]$  allows you to access individual array elements.

Instead you should write:

```
for( i = 1; i < BarCount; i++ )
{
    if ( High[ i ] > High[ i - 1 ] )
    {
        x[ i ] = High[ i ];
    }
    else
    {
        x[ i ] = Low[ i ];
    }
}
```

this will translate to correct one "for EVERY BAR 'i' assign i-th element of high array to the i-th element of x array if i-th element of high array is higher than the previous element, otherwise assign i-th of low array to the i-th element of x array". The rule is: new *if-else* and *while* statements need single boolean value (not array) to decide which execution path should be taken. If you want to use them with arrays you have to iterate through bars using *for* loop (as shown above).

On the other hand this can be implemented in single line using old-style array operations and IIF function:

```
x = IIf( High > Ref( High, -1 ), High, Low );
```

This works because IIF operates on ARRAYS as described in the [tutorial](#).

As you can see in many cases old-style AFL provides much more compact form. I always tried to explain this advantage of AFL but only a few realised that. New control statements should be used where it is better to use them. As I tried to explain during last years in 80% of cases 'old-style' AFL provides the shortest formula. Only remaining 20% of cases needed [script](#). Those 'script-only' cases now can be coded in native AFL thanks to new for/while/if-else statements. And this is correct usage of them – to replace script parts.

## for Statement

The **for** statement lets you repeat a statement or compound statement a specified number of times. The body of a **for** statement is executed zero or more times until an optional condition becomes false.

### Syntax

**for** ( *init-expression* ; *cond-expression* ; *loop-expression* ) *statement*

Execution of a **for** statement proceeds as follows:

1. The *init-expression*, is evaluated. This specifies the initialization for the loop. There is no restriction on the type of *init-expression*.

2. The *cond-expression*, is evaluated. This expression must have arithmetic type. It is evaluated before each iteration. Three results are possible:
  - If *cond-expression* is true (nonzero), *statement* is executed; then *loop-expression*, if any, is evaluated. The *loop-expression* is evaluated after each iteration. There is no restriction on its type. Side effects will execute in order. The process then begins again with the evaluation of *cond-expression*.
  - If *cond-expression* is false (0), execution of the **for** statement terminates and control passes to the next statement in the program.

This example illustrates the **for** statement:

```
myema[ 0 ] = Close[ 0 ];
for( i = 1; i < BarCount; i++ )
{
    myema[ i ] = 0.1 * Close[ i ] + 0.9 * myema[ i - 1 ];
}
```

This example iterates all bars of close array to calculate exponential moving average.

For loop is extremely flexible.

*loop-expression* can be ANY kind of expression you wish. You can produce not only regular series like this:

```
for( i = 0; i < BarCount; i = i + 3 ) // increment by 3 every iteration
```

but you can produce exponential series too:

```
for( i = 1; i < BarCount; i = i * 2 ) // produces series of 1, 2, 4, 8, 16, 32, ...
```

## function Keyword

The **function** keyword begins definition of the user-function.

User-definable functions allow to encapsulate user code into easy-to-use modules that can be user in many places without need to copy the same code over and over again.

Functions must have a definition. The function definition includes the function body the code that executes when the function is called.

A function definition establishes the name, and attributes (or parameters) of a function. A function definition must precede the call to the function. The definition starts with **function** keyword then follows function name, opening parenthesis then optional list of arguments and closing parenthesis. Later comes function body enclosed in curly braces.

A function call passes execution control from the calling function to the called function. The arguments, if any, are passed by value to the called function. Execution of a return statement in the called function returns

control and possibly a value to the calling function.

If the function does not consist of any return statement (does not return anything) then we call it a procedure.

Following is an example of function definition:

```
// the following function is 2nd order smoother

function IIR2( input, f0, f1, f2 )
{
    result[ 0 ] = input[ 0 ];
    result[ 1 ] = input[ 1 ];

    for( i = 2; i < BarCount; i++ )
    {
        result[ i ] = f0 * input[ i ] +
                     f1 * result[ i - 1 ] +
                     f2 * result[ i - 2 ];
    }

    return result;
}

Plot( Close, "Price", colorBlack, styleCandle );
Plot( IIR2( Close, 0.2, 1.4, -0.6 ), "function example", colorRed );
```

In this code **IIR2** is a user-defined function. **input**, **f0**, **f1**, **f2** are formal parameters of the functions.

At the time of function call the values of arguments are passed in these variables. Formal parameters behave like local variables.

Later we have **result** and **i** which are local variables. Local variables are visible inside function only. If any other function uses the same variable name they won't interfere between each other.

## global Keyword

The **global** keyword declares global variable inside user-defined function. Global variable is the variable that is visible/accessible inside the function AND outside the function (at global formula level).

Due to the fact that AFL does not require to declare variables the decision whenever given variable is treated as local or global is taken depends on where it is FIRST USED.

If given identifier appears first INSIDE function definition – then it is treated as LOCAL variable.

If given identifier appears first OUTSIDE function definition – then it is treated as GLOBAL variable.

This default behaviour can be however overridden using **global** and **local** keywords (introduced in 4.36) – see example 2.

*Example (commentary):*

```
k = 4; // this is GLOBAL variable
```

```

function f( x )
{
    z = 3; // this is LOCAL variable
    return z * x * k; // 'k' here references global variable k (first used above
outside function)
}

z = 5; // this is GLOBAL variable with the same name as local variable in
function f

"The value of z before function call :" + WriteVal( z );

// Now even if we call function
// the value of our global variable z
// is not affected by function call because
// global variable z and local variable z are separate and
// arguments are passed by value (not by reference)

"The result of f( z ) = " + WriteVal( f( z ) );

"The value of z after function call is unchanged : " + WriteVal( z );

```

*Example 2: Using local and global keywords to override default visibility rules:*

```

VariableA = 5; // implicit global variable

function Test()
{
    local VariableA; // explicit local variable with the same identifier as
global
    global VariableB; // explicit global variable not defined earlier
                        // may be used to return more than one value from the
function

    VariableA = 99;
    VariableB = 333;
}

VariableB = 1; // global variable

"Before function call";
"VariableA = " + VariableA;
"VariableB = " + VariableB;

Test();

"After function call";
"VariableA = " + VariableA + " (not affected by function call)";
"VariableB = " + VariableB + " (affected by the function call)";

```



## if Keyword

The **if** keyword is an required part of if-else statement.

### if, else Statement

```
if( expression )
    statement1
[else
    statement2]
```

The **if** keyword executes *statement1* if *expression* is true (nonzero); if **else** is present and *expression* is false (zero), it executes *statement2*. After executing *statement1* or *statement2*, control passes to the next statement.

#### Example 1

```
if ( i > 0 )
    y = x / i;
else
{
    x = i;
    y = abs( x );
}
```

In this example, the statement  $y = x/i$  is executed if *i* is greater than 0. If *i* is less than or equal to 0, *i* is assigned to *x* and `abs( x )` is assigned to *y*. Note that the statement forming the **if** clause ends with a semicolon.

When nesting **if** statements and **else** clauses, use braces to group the statements and clauses into compound statements that clarify your intent. If no braces are present, the compiler resolves ambiguities by associating each **else** with the closest **if** that lacks an **else**.

#### Example 2

```
if ( i > 0 )                /* Without braces */
    if ( j > i )
        x = j;
    else
        x = i;
```

The **else** clause is associated with the inner **if** statement in this example. If *i* is less than or equal to 0, no value is assigned to *x*.

#### Example 3

```
if ( i > 0 )
{
    /* With braces */
    if ( j > i )
        x = j;
}
else
    x = i;
```

The braces surrounding the inner **if** statement in this example make the **else** clause part of the outer **if** statement. If `i` is less than or equal to 0, `i` is assigned to `x`.

### Common misunderstandings

*"New if-else problem"*

Question:

Why I get the syntax error when I write: `if( H > Ref(H,-1) )`

Answer:

if-else statement changes flow of execution (opposite to IIF function that evaluates all arguments and works on arrays) and you can not really write

```
if ( H >Ref(H,-1) )
```

because it has no meaning. It would translate to "If high array is higher than high array shifted one bar" (see tutorial below). Flow control statement (such as if-else) has to get SINGLE boolean value to make decision which execution path should be taken. If you write `H` (or `High`) it means ARRAY (entire array).

if you write `H[i]` – it means `i`-th element of the array. The subscript operator `[]` allows you to access individual array elements.

Instead you should write:

```
for( i = 1; i < BarCount; i++ )
{
    if ( High[ i ] > High[ i - 1 ] )
    {
        x[ i ] = High[ i ];
    }
    else
    {
        x[ i ] = Low[ i ];
    }
}
```

this will translate to correct one "for EVERY BAR 'i' assign `i`-th element of high array to the `i`-th element of `x` array if `i`-th element of high array is higher than the previous element, otherwise assign `i`-th of low array to the `i`-th element of `x` array". The rule is: new *if-else* and *while* statements need single boolean value (not array) to decide which execution path should be taken. If you want to use them with arrays you have to iterate through bars using *for* loop (as shown above).

On the other hand this can be implemented in single line using old-style array operations and IIF function:

```
x = IIf( High > Ref( High, -1 ), High, Low );
```

This works because IIF operates on ARRAYS as described in the [tutorial](#).

As you can see in many cases old-style AFL provides much more compact form. I always tried to explain this

advantage of AFL but only a few realised that. New control statements should be used where it is better to use them. As I tried to explain during last years in 80% of cases 'old-style' AFL provides the shortest formula. Only remaining 20% of cases needed [script](#). Those 'script-only' cases now can be coded in native AFL thanks to new for/while/if-else statements. And this is correct usage of them – to replace script parts.

## local Keyword

The **local** keyword declares local variable inside user-defined function. Local variable is the variable that is visible/accessible only inside the function.

Due to the fact that AFL does not require to declare variables the decision whenever given variable is treated as local or global is taken depends on where it is FIRST USED.

If given identifier appears first INSIDE function definition – then it is treated as LOCAL variable.

If given identifier appears first OUTSIDE function definition – then it is treated as GLOBAL variable.

This default behaviour can be however overridden using **global** and **local** keywords (introduced in 4.36) – see example 2.

*Example (commentary):*

```
k = 4; // this is GLOBAL variable

function f( x )
{
    z = 3; // this is LOCAL variable
    return z * x * k; // 'k' here references global variable k (first used above
outside function)
}

z = 5; // this is GLOBAL variable with the same name as local variable in
function f

"The value of z before function call :" + WriteVal( z );

// Now even if we call function
// the value of our global variable z
// is not affected by function call because
// global variable z and local variable z are separate and
// arguments are passed by value (not by reference)

"The result of f( z ) = " + WriteVal( f( z ) );

"The value of z after function call is unchanged : " + WriteVal( z );
```

*Example 2: Using local and global keywords to override default visibility rules:*

```
VariableA = 5; // implicit global variable
```

```

function Test()
{
    local VariableA; // explicit local variable with the same identifier as
    global VariableB; // explicit global variable not defined earlier
    // may be used to return more than one value from the
    function

    VariableA = 99;
    VariableB = 333;
}

VariableB = 1; // global variable

"Before function call";
"VariableA = " + VariableA;
"VariableB = " + VariableB;

Test();

"After function call";
"VariableA = " + VariableA + " (not affected by function call)";
"VariableB = " + VariableB + " (affected by the function call)"

```

## procedure Keyword

The **procedure** keyword begins definition of the user-procedure.

Procedure is a function that does NOT return any value (does not have return statement).

Consult [function keyword](#) help for more details.

## return Keyword

The **return** keyword allows to return the value from the function.

```

function RiseToAPowerOf2( x )
{
    return x ^ 2;
}

```

At the end of the function we can see 'return' statement that is used to return the result to the caller. Note that currently return statement must be placed at the very end of the function.

Consult [function keyword](#) help for more details.

## switch Statement

The switch and case statements help control complex conditional and branching operations. The switch statement transfers control to a statement within its body.

Syntax:

```
switch ( expression )
{

case constant-expression1 : statement;
case constant-expression2 : statement;
...
case constant-expressionN : statement;

default : statement;

}
```

Control passes to the statement whose case *constant-expression* matches the value of switch ( *expression* ). The switch statement can include any number of case instances, but no two case constants within the same switch statement can have the same value. Execution of the statement body begins at the selected statement and proceeds until the end of the body or until a **break** statement transfers control out of the body.

You can use the **break** statement to end processing of a particular case within the switch statement and to branch to the end of the switch statement. Without **break**, the program continues to the next case, executing the statements until a break or the end of the statement is reached. In some situations, this continuation may be desirable.

The **default** statement is executed if no case constant-expression is equal to the value of switch ( *expression* ). If the **default** statement is omitted, and no case match is found, none of the statements in the switch body are executed. There can be at most one default statement. The default statement, if exists, **MUST** come at the end. Otherwise it may be executed before hitting conditions defined below it. A **case** or **default** label is allowed to appear only inside a switch statement.

The type of switch expression and case constant-expression can be any. The value of each case constant-expression must be unique within the statement body. Otherwise first-match will be used.

Example:

```
for( n = 0; n < 10; n++ )
{
    printf("Current n = %f\n", n );

    switch(n) {
        case 0:
            printf("The number is zero.\n");
            break;
        case 3:
        case 5:
        case 7:
```

```

    printf( "n is a prime number\n" );
    break;
case 2: printf( "n is a prime number\n" );
case 4:
case 6:
case 8:
    printf( "n is an even number\n" );
    break;
case 1:
case 9:
    printf( "n is a perfect square\n" );
    break;
default:
    printf( "Only single-digit numbers are allowed\n" );
    break;
}

```

More information can be found here: [http://en.wikipedia.org/wiki/Switch\\_statement](http://en.wikipedia.org/wiki/Switch_statement)

## while Keyword

The **while** keyword is al part of while (described below) and [do-while](#) statements.

## while Statement

The while statement lets you repeat a statement until a specified expression becomes false.

### Syntax

**while** ( *expression* ) *statement*

The *expression* must have arithmetic (numeric/boolean) type. Execution proceeds as follows:

1. The *expression* is evaluated.
2. If *expression* is initially false, the body of the **while** statement is never executed, and control passes from the **while** statement to the next statement in the program.

If *expression* is true (nonzero), the body of the statement is executed and the process is repeated beginning at step 1.

This is an example of the **while** statement:

```

i = 10;
while( i < 20 )
{
    Plot( MA( Close, i ), "MA" + WriteVal( i, 0 ), colorBlack + i );
    i = i + 1;
}

```

The example plots 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 – bar moving averages.

## AFL Function Reference – Categorized list of functions

### Basic price pattern detection

- **FFT** – performs Fast Fourier Transform (AFL 2.90)
- **GAPDOWN** – gap down
- **GAPUP** – gap up
- **INSIDE** – inside day
- **OUTSIDE** – outside bar
- **PEAK** – peak (AFL 1.1)
- **PEAKBARS** – bars since peak (AFL 1.1)
- **TROUGH** – trough (AFL 1.1)
- **TROUGHBARS** – bars since trough (AFL 1.1)
- **ZIG** – zig-zag indicator (AFL 1.1)

### Composites

- **AddToComposite** – add value to composite ticker (AFL 2.0)
- **ADLine** – advance/decline line (AFL 1.2)
- **AdvIssues** – advancing issues (AFL 1.2)
- **AdvVolume** – advancing issues volume (AFL 1.2)
- **DeclIssues** – declining issues (AFL 1.2)
- **DeclVolume** – declining issues volume (AFL 1.2)
- **TRIN** – traders (Arms) index (AFL 1.2)
- **UNCISSUES** – unchanged issues (AFL 1.2)
- **UNCVOLUME** – unchanged issues volume (AFL 1.2)

### Date/Time

- **BarIndex** – get zero-based bar number (AFL 2.3)
- **BeginValue** – Value of the array at the begin of the range (AFL 2.3)
- **Date** – date (AFL 1.1)
- **DateNum** – date number (AFL 1.4)
- **DateTime** – retrieves encoded date time (AFL 2.3)
- **DateTimeConvert** – date/time format conversion (AFL 2.90)
- **Day** – day of month (AFL 1.4)
- **DayOfWeek** – day of week (AFL 1.4)
- **DayOfYear** – get ordinal number of day in a year (AFL 2.4)
- **EndValue** – value of the array at the end of the selected range (AFL 2.3)
- **GetPlaybackDateTime** – get bar replay position date/time (AFL 3.0)
- **HOURL** – get current bar's hour (AFL 2.0)
- **INTERVAL** – get bar interval (in seconds) (AFL 2.1)
- **MINUTE** – get current bar's minute (AFL 2.0)
- **MONTH** – month (AFL 1.4)
- **NOW** – gets current system date/time (AFL 2.3)
- **SECOND** – get current bar's second (AFL 2.0)
- **TIMENUM** – get current bar time (AFL 2.0)

- **YEAR** – year (AFL 1.4)

## Indicators

- **AccDist** – accumulation/distribution
- **ADX** – average directional movement index (AFL 1.3)
- **ATR** – average true range (AFL 1.3)
- **BBandBot** – bottom bollinger band
- **BBandTop** – top bollinger band
- **CCI** – commodity channel index
- **Chaikin** – chaikin oscillator
- **GetCursorMouseButtons** – get current state of mouse buttons (AFL 2.80)
- **GetCursorXPosition** – get current X position of mouse pointer (AFL 2.80)
- **GetCursorYPosition** – get current Y position of mouse pointer (AFL 2.80)
- **MACD** – moving average convergence/divergence
- **MDI** – minus directional movement indicator (–DI) (AFL 1.3)
- **MFI** – money flow index
- **NVI** – negative volume index
- **OBV** – on balance volume
- **OSCP** – price oscillator
- **OSCV** – volume oscillator
- **PDI** – plus directional movement indicator (AFL 1.3)
- **PlotText** – write text on the chart (AFL 2.80)
- **PVI** – positive volume index
- **RequestTimedRefresh** – forces periodical refresh of indicator pane (AFL 2.90)
- **RMI** – Relative Momentum Index (AFL 2.1)
- **ROC** – percentage rate of change
- **RSI** – relative strength index
- **RWI** – random walk index
- **RWIHI** – random walk index of highs
- **RWILO** – random walk index of lows
- **SAR** – parabolic stop–and–reverse (AFL 1.3)
- **SetChartBkColor** – set background color of a chart (AFL 2.80)
- **SetChartBkGradientFill** – enables background gradient color fill in indicators (AFL 2.90)
- **SIGNAL** – macd signal line
- **STOCHD** – stochastic slow %D
- **STOCHK** – stochastic slow %K
- **TRIX** – triple exponential smoothed price
- **ULTIMATE** – ultimate oscillator

## Information / Categories

- **CategoryAddSymbol** – adds a symbol to a category (AFL 2.5)
- **CategoryFind** – search for category by name (AFL 3.0)
- **CategoryGetName** – get the name of a category (AFL 2.5)
- **CategoryGetSymbols** – retrieves comma-separated list of symbols belonging to given category (AFL 2.5)
- **CategoryRemoveSymbol** – remove a symbol from a category (AFL 2.5)
- **FullName** – full name of the symbol (AFL 1.1)
- **GetCategorySymbols** – retrieves comma-separated list of symbols belonging to given category (AFL 2.4)
- **GetDatabaseName** – retrieves folder name of current database (AFL 2.3)



- **GetFnData** – get fundamental data (AFL 2.90)
- **GROUPLD** – get group ID/name (AFL 1.8)
- **INDUSTRYID** – get industry ID / name (AFL 1.8)
- **InWatchList** – watch list membership test (by ordinal number)
- **InWatchListName** – watch list membership test (by name) (AFL 3.0)
- **IsContinuous** – checks 'continuous quotations' flag state (AFL 2.60)
- **IsFavorite** – check if current symbol belongs to favorites (AFL 2.5)
- **IsIndex** – check if current symbol is an index (AFL 2.5)
- **MARKETID** – market ID / name (AFL 1.8)
- **NAME** – ticker symbol (AFL 1.1)
- **SECTORID** – get sector ID / name (AFL 1.8)

### Lowest/Highest

- **HHV** – highest high value
- **HHVBARS** – bars since highest high
- **HIGHEST** – highest value
- **HIGHESTBARS** – bars since highest value
- **HIGHESTSINCE** – highest value since condition met (AFL 1.4)
- **HIGHESTSINCEBARS** – bars since highest value since condition met (AFL 1.4)
- **LLV** – lowest low value
- **LLVBARS** – bars since lowest low
- **LOWEST** – lowest value
- **LOWESTBARS** – bars since lowest
- **LOWESTSINCE** – lowest value since condition met (AFL 1.4)
- **LOWESTSINCEBARS** – bars since lowest value since condition met (AFL 1.4)

### Math functions

- **abs** – absolute value
- **acos** – arccosine function
- **AlmostEqual** – rounding error insensitive comparison (AFL 2.80)
- **asin** – arcsine function
- **atan** – arc tan
- **atan2** – calculates arctangent of y/x (AFL 2.90)
- **ceil** – ceil value
- **cos** – cosine
- **cosh** – hyperbolic cosine function (AFL 2.80)
- **EXP** – exponential function
- **FLOOR** – floor value
- **FRAC** – fractional part
- **INT** – integer part
- **LOG** – natural logarithm
- **LOG10** – decimal logarithm
- **MAX** – maximum value of two numbers / arrays
- **MIN** – minimum value of two numbers / arrays
- **PREC** – adjust number of decimal points of floating point number
- **ROUND** – round number to nearest integer
- **sign** – returns the sign of the number/array (AFL 2.50)
- **SIN** – sine function
- **sinh** – hyperbolic sine function (AFL 2.80)
- **SQRT** – square root

- **tan** – tangent function (AFL 1.0)
- **tanh** – hyperbolic tangent function (AFL 2.80)

### Miscellaneous functions

- **#include** – preprocessor include command (AFL 2.2)
- **#include\_once** – preprocessor include (once) command (AFL 2.70)
- **#pragma** – sets AFL pre-processor option (AFL 2.4)
- **ClipboardGet** – retrieves current contents of Windows clipboard (AFL 2.60)
- **ClipboardSet** – copies the text to the Windows clipboard (AFL 2.6)
- **ColorHSB** – specify color using Hue–Saturation–Brightness (AFL 2.80)
- **ColorRGB** – specify color using Red–Green–Blue components (AFL 2.80)
- **CreateObject** – create COM object (AFL 1.8)
- **CreateStaticObject** – create static COM object (AFL 1.8)
- **EnableScript** – enable scripting engine
- **EnableTextOutput** – enables/disables text output in the Chart Commentary window (AFL 2.2)
- **GETEXTRADATA** – get extra data from external data source (AFL 1.9)
- **GetPerformanceCounter** – retrieves the current value of the high–resolution performance counter (AFL 2.90)
- **GetRTData** – retrieves the real–time data fields (AFL 2.60)
- **GetRTDataForeign** – retrieves the real–time data fields (for specified symbol) (AFL 2.80)
- **GETSCRIPTOBJECT** – get access to script COM object (AFL 1.8)
- **ISEMPTY** – empty value check (AFL 1.5)
- **ISFINITE** – check if value is not infinite (AFL 2.3)
- **ISNAN** – checks for NaN (not a number) (AFL 2.3)
- **ISNULL** – check for Null (empty) value (AFL 2.3)
- **ISTRUE** – true value (non–empty and non–zero) check (AFL 1.5)
- **NoteGet** – retrieves the text of the note (AFL 2.6)
- **NoteSet** – sets text of the note (AFL 2.6)
- **NZ** – Null (Null/Nan/Infinity) to zero (AFL 2.3)
- **PopupWindow** – display pop–up window (AFL 3.0)
- **PREFS** – retrieve preferences settings (AFL 1.4)
- **Say** – speaks provided text (AFL 2.90)
- **SETBARSREQUIRED** – set number of previous and future bars needed for script/DLL to properly execute (AFL 2.1)
- **StaticVarGet** – gets the value of static variable (AFL 2.60)
- **StaticVarGetText** – gets the value of static variable as string (AFL 2.60)
- **StaticVarRemove** – remove static variable (AFL 2.80)
- **StaticVarSet** – sets the value of static variable (AFL 2.60)
- **StaticVarSetText** – Sets the value of static string variable. (AFL 2.60)
- **STATUS** – get run–time AFL status information (AFL 1.65)
- **STUDY** – reference hand–drawn study (AFL 1.5)
- **VarGet** – gets the value of dynamic variable (AFL 2.60)
- **VarGetText** – gets the text value of dynamic variable (AFL 2.80)
- **VarSet** – sets the value of dynamic variable (AFL 2.60)
- **VarSetText** – sets dynamic variable of string type (AFL 2.80)
- **VERSION** – get version info (AFL 1.9)
- **\_TRACE** – print text to system debug viewer (AFL 2.4)

### Moving averages, summation

- **AMA** – adaptive moving average (AFL 1.5)

- **AMA2** – adaptive moving average (AFL 1.5)
- **Cum** – cumulative sum
- **DEMA** – double exponential moving average (AFL 2.0)
- **EMA** – exponential moving average
- **MA** – simple moving average
- **SUM** – sum data over specified number of bars
- **TEMA** – triple exponential moving average (AFL 2.0)
- **WILDERS** – Wilder's smoothing (AFL 1.4)
- **WMA** – weighted moving average (AFL 2.0)

### Statistical functions

- **Correlation** – correlation (AFL 1.4)
- **LINEARREG** – linear regression end-point (AFL 2.2)
- **LINREGINTERCEPT** – (AFL 2.2)
- **LINREGSLOPE** – linear regression slope (AFL 1.4)
- **Median** – calculate median (middle element) (AFL 2.5)
- **mtRandom** – Mersene Twister random number generator (AFL 3.0)
- **Percentile** – calculate percentile (AFL 2.5)
- **RANDOM** – random number (AFL 1.9)
- **STDERR** – standard error (AFL 1.4)
- **STDEV** – standard deviation (AFL 1.4)
- **TSF** – time series forecast (AFL 2.2)

### String manipulation

- **Asc** – get ASCII code of character (AFL 2.80)
- **DateTimeToStr** – convert datetime to string (AFL 2.8)
- **NumToStr** – convert number to string (AFL 2.5)
- **printf** – Print formatted output to the output window. (AFL 2.5)
- **StrExtract** – extracts given item (substring) from comma-separated string (AFL 2.4)
- **StrFind** – find substring in a string (AFL 2.5)
- **StrFormat** – Write formatted output to the string (AFL 2.5)
- **StrLeft** – extracts the leftmost part (AFL 2.0)
- **StrLen** – string length (AFL 1.5)
- **StrMid** – extracts part of the string (AFL 2.0)
- **StrReplace** – string replace (AFL 2.90)
- **StrRight** – extracts the rightmost part of the string (AFL 2.0)
- **StrToDateTime** – convert string to datetime (AFL 2.80)
- **StrToLower** – convert to lowercase (AFL 2.80)
- **StrToNum** – convert string to number (AFL 2.5)
- **StrToUpper** – convert to uppercase (AFL 2.80)

### Trading system toolbox

- **AlertIf** – trigger alerts (AFL 2.1)
- **ApplyStop** – apply built-in stop (AFL 1.7)
- **BarsSince** – bars since
- **Cross** – crossover check
- **EnableRotationalTrading** – Turns on rotational-trading mode of the backtester (AFL 2.5)
- **Equity** – calculate single-symbol equity line (AFL 2.0)
- **EXREM** – remove excessive signals (AFL 1.5)

- **EXREMSPAN** – remove excessive signals spanning given number of bars (AFL 2.0)
- **FLIP** – (AFL 1.5)
- **GetOption** – gets the value of option in automatic analysis settings (AFL 2.60)
- **GetTradingInterface** – retrieves OLE automation object to automatic trading interfac (AFL 2.70)
- **HOLD** – hold the alert signal
- **IIF** – immediate IF function
- **LASTVALUE** – last value of the array
- **OPTIMIZE** – define optimization variable (AFL 1.7)
- **REF** – reference past/future values of the array
- **SetBacktestMode** – Sets working mode of the backtester (AFL 3.0)
- **SetCustomBacktestProc** – define custom backtest procedure formula file (AFL 2.70)
- **SetFormulaName** – set the name of the formula (AFL 2.5)
- **SetOption** – sets options in automatic analysis settings (AFL 2.3)
- **SetPositionSize** – set trade size (AFL 2.70)
- **SETTRADEDELAYS** – allows to control trade delays applied by the backtester (AFL 2.1)
- **VALUEWHEN** – get value of the array when condition met (AFL 1.1)

## Exploration / Indicators

- **AddColumn** – add numeric exploration column (AFL 1.8)
- **AddTextColumn** – add text exploration column (AFL 1.8)
- **EncodeColor** – encodes color for indicator title (AFL 2.2)
- **GETCHARTID** – get current chart ID (AFL 2.3)
- **GetPriceStyle** – get current price chart style (AFL 2.70)
- **LineArray** – generate trend–line array (AFL 2.5)
- **PARAM** – add user user–definable numeric parameter (AFL 2.3)
- **ParamColor** – add user user–definable color parameter (AFL 2.3)
- **ParamDate** – add user user–definable date parameter (AFL 2.60)
- **ParamField** – creates price field parameter (AFL 2.70)
- **ParamList** – creates the parameter that consist of the list of choices (AFL 2.70)
- **PARAMSTR** – add user user–definable string parameter (AFL 2.3)
- **ParamStyle** – select styles applied to the plot (AFL 2.70)
- **ParamTime** – add user user–definable time parameter (AFL 2.60)
- **ParamToggle** – create Yes/No parameter (AFL 2.70)
- **ParamTrigger** – creates a trigger (button) in the parameter dialog (AFL 2.70)
- **PLOT** – plot indicator graph (AFL 1.8)
- **PLOTGRID** – Plot horizontal grid line (AFL 2.3)
- **PLOTOHLC** – plot custom OHLC chart (AFL 2.2)
- **PLOTSHAPES** – plots arrows and other shapes (AFL 2.3)
- **PLOTVAPOVERLAY** – plot Volume–At–Price overlay chart (AFL 2.4)
- **SELECTEDVALUE** – retrieves value of the array at currently selected date/time point (AFL 2.1)
- **SetChartOptions** – set/clear/overwrite defaults for chart pane options (AFL 2.70)
- **SetSortColumns** – sets the columns which will be used for sorting in AA window (AFL 2.90)
- **WRITEIF** – commentary conditional text output
- **WriteVal** – write number or value of the array
- **\_DEFAULT\_NAME** – retrieve default name of the plot (AFL 2.70)
- **\_N** – no text output (AFL 2.1)
- **\_PARAM\_VALUES** – retrieve param values string (AFL 2.70)
- **\_SECTION\_BEGIN** – section begin marker (AFL 2.70)
- **\_SECTION\_END** – section end marker (AFL 2.70)
- **\_SECTION\_NAME** – retrieve current section name (AFL 2.70)

**File Input/Output functions**

- **fclose** – close a file (AFL 2.5)
- **fdelete** – deletes a file (AFL 2.70)
- **feof** – test for end-of-file (AFL 2.5)
- **fgets** – get a string from a file (AFL 2.5)
- **fgetstatus** – retrieves file status/properties (AFL 2.90)
- **fmkdir** – creates (makes) a directory (AFL 2.70)
- **fopen** – open a file (AFL 2.5)
- **fputs** – write a string to a file (AFL 2.5)
- **frmdir** – removes a directory (AFL 2.70)

**Low-level graphics**

- **GfxArc** – draw an arc (AFL 3.0)
- **GfxChord** – draw a chord (AFL 3.0)
- **GfxCircle** – draw a circle (AFL 3.0)
- **GfxDrawText** – draw a text (clipped to rectangle) (AFL 3.0)
- **GfxEllipse** – draw an ellipse (AFL 3.0)
- **GfxGradientRect** – draw a rectangle with gradient fill (AFL 3.0)
- **GfxLineTo** – draw a line to specified point (AFL 3.0)
- **GfxMoveTo** – move graphic cursor to new position (AFL 3.0)
- **GfxPie** – draw a pie (AFL 3.0)
- **GfxPolygon** – draw a polygon (AFL 3.0)
- **GfxPolyline** – draw a polyline (AFL 3.0)
- **GfxRectangle** – draw a rectangle (AFL 3.0)
- **GfxRoundRect** – draw a rectangle with rounded corners (AFL 3.0)
- **GfxSelectFont** – create / select graphic font (AFL 3.0)
- **GfxSelectPen** – create / select graphic pen (AFL 3.0)
- **GfxSelectSolidBrush** – create / select graphic brush (AFL 3.0)
- **GfxSetBkColor** – set graphic background color (AFL 3.0)
- **GfxSetBkMode** – set graphic background mode (AFL 3.0)
- **GfxSetOverlayMode** – set low-level graphic overlay mode (AFL 3.0)
- **GfxSetPixel** – set pixel at specified position to specified color (AFL 3.0)
- **GfxSetTextAlign** – set text alignment (AFL 3.0)
- **GfxSetTextColor** – set graphic text color (AFL 3.0)
- **GfxTextOut** – writes text at the specified location (AFL 3.0)

**Referencing other symbol data**

- **FOREIGN** – access foreign security data (AFL 1.5)
- **GETBASEINDEX** – retrieves symbol of relative strength base index (AFL 2.1)
- **PLOTFOREIGN** – plot foreign security data (AFL 2.2)
- **RELSTRENGTH** – comparative relative strength (AFL 1.3)
- **RestorePriceArrays** – restore price arrays to original symbol (AFL 2.5)
- **SetForeign** – replace current price arrays with those of foreign security (AFL 2.5)

**Time Frame functions**

- **TimeFrameCompress** – compress single array to given time frame (AFL 2.5)
- **TimeFrameExpand** – expand time frame compressed array (AFL 2.5)
- **TimeFrameGetPrice** – retrieve O, H, L, C, V values from other time frame (AFL 2.5)

- **TimeFrameMode** – switch time frame compression mode (AFL 2.80)
- **TimeFrameRestore** – restores price arrays to original time frame (AFL 2.5)
- **TimeFrameSet** – switch price arrays to a different time frame (AFL 2.5)

## AFL Function Reference – Alphabetical list of functions

1. **#include** ( Miscellaneous functions) – preprocessor include command (AFL 2.2)
2. **#include\_once** ( Miscellaneous functions) – preprocessor include (once) command (AFL 2.70)
3. **#pragma** ( Miscellaneous functions) – sets AFL pre-processor option (AFL 2.4)
4. **abs** ( Math functions) – absolute value
5. **AccDist** ( Indicators) – accumulation/distribution
6. **acos** ( Math functions) – arccosine function
7. **AddColumn** (Exploration / Indicators) – add numeric exploration column (AFL 1.8)
8. **AddTextColumn** (Exploration / Indicators) – add text exploration column (AFL 1.8)
9. **AddToComposite** ( Composites) – add value to composite ticker (AFL 2.0)
10. **ADLine** ( Composites) – advance/decline line (AFL 1.2)
11. **AdvIssues** ( Composites) – advancing issues (AFL 1.2)
12. **AdvVolume** ( Composites) – advancing issues volume (AFL 1.2)
13. **ADX** ( Indicators) – average directional movement index (AFL 1.3)
14. **AlertIf** ( Trading system toolbox) – trigger alerts (AFL 2.1)
15. **AlmostEqual** ( Math functions) – rounding error insensitive comparison (AFL 2.80)
16. **AMA** ( Moving averages, summation) – adaptive moving average (AFL 1.5)
17. **AMA2** ( Moving averages, summation) – adaptive moving average (AFL 1.5)
18. **ApplyStop** ( Trading system toolbox) – apply built-in stop (AFL 1.7)
19. **Asc** ( String manipulation) – get ASCII code of character (AFL 2.80)
20. **asin** ( Math functions) – arcsine function
21. **atan** ( Math functions) – arc tan
22. **atan2** ( Math functions) – calculates arctangent of y/x (AFL 2.90)
23. **ATR** ( Indicators) – average true range (AFL 1.3)
24. **BarIndex** ( Date/Time) – get zero-based bar number (AFL 2.3)
25. **BarsSince** ( Trading system toolbox) – bars since
26. **BBandBot** ( Indicators) – bottom bollinger band
27. **BBandTop** ( Indicators) – top bollinger band
28. **BeginValue** ( Date/Time) – Value of the array at the begin of the range (AFL 2.3)
29. **CategoryAddSymbol** ( Information / Categories) – adds a symbol to a category (AFL 2.5)
30. **CategoryFind** ( Information / Categories) – search for category by name (AFL 3.0)
31. **CategoryGetName** ( Information / Categories) – get the name of a category (AFL 2.5)
32. **CategoryGetSymbols** ( Information / Categories) – retrieves comma-separated list of symbols belonging to given category (AFL 2.5)
33. **CategoryRemoveSymbol** ( Information / Categories) – remove a symbol from a category (AFL 2.5)
34. **CCI** ( Indicators) – commodity channel index
35. **ceil** ( Math functions) – ceil value
36. **Chaikin** ( Indicators) – chaikin oscillator
37. **ClipboardGet** ( Miscellaneous functions) – retrieves current contents of Windows clipboard (AFL 2.60)
38. **ClipboardSet** ( Miscellaneous functions) – copies the text to the Windows clipboard (AFL 2.6)
39. **ColorHSB** ( Miscellaneous functions) – specify color using Hue-Saturation-Brightness (AFL 2.80)
40. **ColorRGB** ( Miscellaneous functions) – specify color using Red-Green-Blue components (AFL 2.80)
41. **Correlation** ( Statistical functions) – correlation (AFL 1.4)
42. **cos** ( Math functions) – cosine
43. **cosh** ( Math functions) – hyperbolic cosine function (AFL 2.80)



44. **CreateObject** ( Miscellaneous functions) – create COM object (AFL 1.8)
45. **CreateStaticObject** ( Miscellaneous functions) – create static COM object (AFL 1.8)
46. **Cross** ( Trading system toolbox) – crossover check
47. **Cum** ( Moving averages, summation) – cumulative sum
48. **Date** ( Date/Time) – date (AFL 1.1)
49. **DateNum** ( Date/Time) – date number (AFL 1.4)
50. **DateTime** ( Date/Time) – retrieves encoded date time (AFL 2.3)
51. **DateTimeConvert** ( Date/Time) – date/time format conversion (AFL 2.90)
52. **DateTimeToStr** ( String manipulation) – convert datetime to string (AFL 2.8)
53. **Day** ( Date/Time) – day of month (AFL 1.4)
54. **DayOfWeek** ( Date/Time) – day of week (AFL 1.4)
55. **DayOfYear** ( Date/Time) – get ordinal number of day in a year (AFL 2.4)
56. **DeclIssues** ( Composites) – declining issues (AFL 1.2)
57. **DeclVolume** ( Composites) – declining issues volume (AFL 1.2)
58. **DEMA** ( Moving averages, summation) – double exponential moving average (AFL 2.0)
59. **EMA** ( Moving averages, summation) – exponential moving average
60. **EnableRotationalTrading** ( Trading system toolbox) – Turns on rotational–trading mode of the backtester (AFL 2.5)
61. **EnableScript** ( Miscellaneous functions) – enable scripting engine
62. **EnableTextOutput** ( Miscellaneous functions) – enables/disables text output in the Chart Commentary window (AFL 2.2)
63. **EncodeColor** (Exploration / Indicators) – encodes color for indicator title (AFL 2.2)
64. **EndValue** ( Date/Time) – value of the array at the end of the selected range (AFL 2.3)
65. **Equity** ( Trading system toolbox) – calculate single–symbol equity line (AFL 2.0)
66. **EXP** ( Math functions) – exponential function
67. **EXREM** ( Trading system toolbox) – remove excessive signals (AFL 1.5)
68. **EXREMSPAN** ( Trading system toolbox) – remove excessive signals spanning given number of bars (AFL 2.0)
69. **fclose** (File Input/Output functions) – close a file (AFL 2.5)
70. **fdelete** (File Input/Output functions) – deletes a file (AFL 2.70)
71. **feof** (File Input/Output functions) – test for end–of–file (AFL 2.5)
72. **FFT** ( Basic price pattern detection) – performs Fast Fourier Transform (AFL 2.90)
73. **fgets** (File Input/Output functions) – get a string from a file (AFL 2.5)
74. **fgetstatus** (File Input/Output functions) – retrieves file status/properties (AFL 2.90)
75. **FLIP** ( Trading system toolbox) – (AFL 1.5)
76. **FLOOR** ( Math functions) – floor value
77. **fmkdir** (File Input/Output functions) – creates (makes) a directory (AFL 2.70)
78. **fopen** (File Input/Output functions) – open a file (AFL 2.5)
79. **FOREIGN** (Referencing other symbol data) – access foreign security data (AFL 1.5)
80. **fputs** (File Input/Output functions) – write a string to a file (AFL 2.5)
81. **FRAC** ( Math functions) – fractional part
82. **frmdir** (File Input/Output functions) – removes a directory (AFL 2.70)
83. **FullName** ( Information / Categories) – full name of the symbol (AFL 1.1)
84. **GAPDOWN** ( Basic price pattern detection) – gap down
85. **GAPUP** ( Basic price pattern detection) – gap up
86. **GETBASEINDEX** (Referencing other symbol data) – retrieves symbol of relative strength base index (AFL 2.1)
87. **GetCategorySymbols** ( Information / Categories) – retrieves comma–separated list of symbols belonging to given category (AFL 2.4)
88. **GETCHARTID** (Exploration / Indicators) – get current chart ID (AFL 2.3)
89. **GetCursorMouseButtons** ( Indicators) – get current state of mouse buttons (AFL 2.80)
90. **GetCursorXPosition** ( Indicators) – get current X position of mouse pointer (AFL 2.80)

91. **GetCursorYPosition** ( Indicators ) – get current Y position of mouse pointer (AFL 2.80)
92. **GetDatabaseName** ( Information / Categories ) – retrieves folder name of current database (AFL 2.3)
93. **GETEXTRADATA** ( Miscellaneous functions ) – get extra data from external data source (AFL 1.9)
94. **GetFnData** ( Information / Categories ) – get fundamental data (AFL 2.90)
95. **GetOption** ( Trading system toolbox ) – gets the value of option in automatic analysis settings (AFL 2.60)
96. **GetPerformanceCounter** ( Miscellaneous functions ) – retrieves the current value of the high-resolution performance counter (AFL 2.90)
97. **GetPlaybackDateTime** ( Date/Time ) – get bar replay position date/time (AFL 3.0)
98. **GetPriceStyle** ( Exploration / Indicators ) – get current price chart style (AFL 2.70)
99. **GetRTData** ( Miscellaneous functions ) – retrieves the real-time data fields (AFL 2.60)
100. **GetRTDataForeign** ( Miscellaneous functions ) – retrieves the real-time data fields (for specified symbol) (AFL 2.80)
101. **GETSCRIPTOBJECT** ( Miscellaneous functions ) – get access to script COM object (AFL 1.8)
102. **GetTradingInterface** ( Trading system toolbox ) – retrieves OLE automation object to automatic trading interfac (AFL 2.70)
103. **GfxArc** (Low-level graphics) – draw an arc (AFL 3.0)
104. **GfxChord** (Low-level graphics) – draw a chord (AFL 3.0)
105. **GfxCircle** (Low-level graphics) – draw a circle (AFL 3.0)
106. **GfxDrawText** (Low-level graphics) – draw a text (clipped to rectangle) (AFL 3.0)
107. **GfxEllipse** ( Basic price pattern detection ) – draw an ellipse (AFL 3.0)
108. **GfxGradientRect** (Low-level graphics) – draw a rectangle with gradient fill (AFL 3.0)
109. **GfxLineTo** (Low-level graphics) – draw a line to specified point (AFL 3.0)
110. **GfxMoveTo** (Low-level graphics) – move graphic cursor to new position (AFL 3.0)
111. **GfxPie** (Low-level graphics) – draw a pie (AFL 3.0)
112. **GfxPolygon** (Low-level graphics) – draw a polygon (AFL 3.0)
113. **GfxPolyline** (Low-level graphics) – draw a polyline (AFL 3.0)
114. **GfxRectangle** (Low-level graphics) – draw a rectangle (AFL 3.0)
115. **GfxRoundRect** (Low-level graphics) – draw a rectangle with rounded corners (AFL 3.0)
116. **GfxSelectFont** (Low-level graphics) – create / select graphic font (AFL 3.0)
117. **GfxSelectPen** (Low-level graphics) – create / select graphic pen (AFL 3.0)
118. **GfxSelectSolidBrush** (Low-level graphics) – create / select graphic brush (AFL 3.0)
119. **GfxSetBkColor** (Low-level graphics) – set graphic background color (AFL 3.0)
120. **GfxSetBkMode** (Low-level graphics) – set graphic background mode (AFL 3.0)
121. **GfxSetOverlayMode** (Low-level graphics) – set low-level graphic overlay mode (AFL 3.0)
122. **GfxSetPixel** (Low-level graphics) – set pixel at specified position to specified color (AFL 3.0)
123. **GfxSetTextAlign** (Low-level graphics) – set text alignment (AFL 3.0)
124. **GfxSetTextColor** (Low-level graphics) – set graphic text color (AFL 3.0)
125. **GfxTextOut** (Low-level graphics) – writes text at the specified location (AFL 3.0)
126. **GROUPID** ( Information / Categories ) – get group ID/name (AFL 1.8)
127. **HHV** ( Lowest/Highest ) – highest high value
128. **HHVBARS** ( Lowest/Highest ) – bars since highest high
129. **HIGHEST** ( Lowest/Highest ) – highest value
130. **HIGHESTBARS** ( Lowest/Highest ) – bars since highest value
131. **HIGHESTSINCE** ( Lowest/Highest ) – highest value since condition met (AFL 1.4)
132. **HIGHESTSINCEBARS** ( Lowest/Highest ) – bars since highest value since condition met (AFL 1.4)
133. **HOLD** ( Trading system toolbox ) – hold the alert signal
134. **HOURL** ( Date/Time ) – get current bar's hour (AFL 2.0)
135. **IIF** ( Trading system toolbox ) – immediate IF function
136. **INDUSTRYID** ( Information / Categories ) – get industry ID / name (AFL 1.8)
137. **INSIDE** ( Basic price pattern detection ) – inside day
138. **INT** ( Math functions ) – integer part



139. **INTERVAL** ( Date/Time) – get bar interval (in seconds) (AFL 2.1)
140. **InWatchList** ( Information / Categories) – watch list membership test (by ordinal number)
141. **InWatchListName** ( Information / Categories) – watch list membership test (by name) (AFL 3.0)
142. **IsContinuous** ( Information / Categories) – checks 'continuous quotations' flag state (AFL 2.60)
143. **ISEMPTY** ( Miscellaneous functions) – empty value check (AFL 1.5)
144. **IsFavorite** ( Information / Categories) – check if current symbol belongs to favorites (AFL 2.5)
145. **ISFINITE** ( Miscellaneous functions) – check if value is not infinite (AFL 2.3)
146. **IsIndex** ( Information / Categories) – check if current symbol is an index (AFL 2.5)
147. **ISNAN** ( Miscellaneous functions) – checks for NaN (not a number) (AFL 2.3)
148. **ISNULL** ( Miscellaneous functions) – check for Null (empty) value (AFL 2.3)
149. **ISTRUE** ( Miscellaneous functions) – true value (non-empty and non-zero) check (AFL 1.5)
150. **LASTVALUE** ( Trading system toolbox) – last value of the array
151. **LineArray** ( Exploration / Indicators) – generate trend-line array (AFL 2.5)
152. **LINEARREG** ( Statistical functions) – linear regression end-point (AFL 2.2)
153. **LINREGINTERCEPT** ( Statistical functions) – (AFL 2.2)
154. **LINREGSLOPE** ( Statistical functions) – linear regression slope (AFL 1.4)
155. **LLV** ( Lowest/Highest) – lowest low value
156. **LLVBARS** ( Lowest/Highest) – bars since lowest low
157. **LOG** ( Math functions) – natural logarithm
158. **LOG10** ( Math functions) – decimal logarithm
159. **LOWEST** ( Lowest/Highest) – lowest value
160. **LOWESTBARS** ( Lowest/Highest) – bars since lowest
161. **LOWESTSINCE** ( Lowest/Highest) – lowest value since condition met (AFL 1.4)
162. **LOWESTSINCEBARS** ( Lowest/Highest) – barssince lowest value since condition met (AFL 1.4)
163. **MA** ( Moving averages, summation) – simple moving average
164. **MACD** ( Indicators) – moving average convergence/divergence
165. **MARKETID** ( Information / Categories) – market ID / name (AFL 1.8)
166. **MAX** ( Math functions) – maximum value of two numbers / arrays
167. **MDI** ( Indicators) – minus directional movement indicator (–DI) (AFL 1.3)
168. **Median** ( Statistical functions) – calculate median (middle element) (AFL 2.5)
169. **MFI** ( Indicators) – money flow index
170. **MIN** ( Math functions) – minimum value of two numbers / arrays
171. **MINUTE** ( Date/Time) – get current bar's minute (AFL 2.0)
172. **MONTH** ( Date/Time) – month (AFL 1.4)
173. **mtRandom** ( Statistical functions) – Mersene Twister random number generator (AFL 3.0)
174. **NAME** ( Information / Categories) – ticker symbol (AFL 1.1)
175. **NoteGet** ( Miscellaneous functions) – retrieves the text of the note (AFL 2.6)
176. **NoteSet** ( Miscellaneous functions) – sets text of the note (AFL 2.6)
177. **NOW** ( Date/Time) – gets current system date/time (AFL 2.3)
178. **NumToStr** ( String manipulation) – convert number to string (AFL 2.5)
179. **NVI** ( Indicators) – negative volume index
180. **NZ** ( Miscellaneous functions) – Null (Null/Nan/Infinity) to zero (AFL 2.3)
181. **OBV** ( Indicators) – on balance volume
182. **OPTIMIZE** ( Trading system toolbox) – define optimization variable (AFL 1.7)
183. **OSCP** ( Indicators) – price oscillator
184. **OSCV** ( Indicators) – volume oscillator
185. **OUTSIDE** ( Basic price pattern detection) – outside bar
186. **PARAM** ( Exploration / Indicators) – add user user-definable numeric parameter (AFL 2.3)
187. **ParamColor** ( Exploration / Indicators) – add user user-definable color parameter (AFL 2.3)
188. **ParamDate** ( Exploration / Indicators) – add user user-definable date parameter (AFL 2.60)
189. **ParamField** ( Exploration / Indicators) – creates price field parameter (AFL 2.70)

190. **ParamList** (Exploration / Indicators) – creates the parameter that consist of the list of choices (AFL 2.70)
191. **PARAMSTR** (Exploration / Indicators) – add user user–definable string parameter (AFL 2.3)
192. **ParamStyle** (Exploration / Indicators) – select styles applied to the plot (AFL 2.70)
193. **ParamTime** (Exploration / Indicators) – add user user–definable time parameter (AFL 2.60)
194. **ParamToggle** (Exploration / Indicators) – create Yes/No parameter (AFL 2.70)
195. **ParamTrigger** (Exploration / Indicators) – creates a trigger (button) in the parameter dialog (AFL 2.70)
196. **PD** ( Indicators) – plus directional movement indicator (AFL 1.3)
197. **PEAK** ( Basic price pattern detection) – peak (AFL 1.1)
198. **PEAKBARS** ( Basic price pattern detection) – bars since peak (AFL 1.1)
199. **Percentile** ( Statistical functions) – calculate percentile (AFL 2.5)
200. **PLOT** (Exploration / Indicators) – plot indicator graph (AFL 1.8)
201. **PLOTFOREIGN** (Referencing other symbol data) – plot foreign security data (AFL 2.2)
202. **PLOTGRID** (Exploration / Indicators) – Plot horizontal grid line (AFL 2.3)
203. **PLOTOHLC** (Exploration / Indicators) – plot custom OHLC chart (AFL 2.2)
204. **PLOTSHAPES** (Exploration / Indicators) – plots arrows and other shapes (AFL 2.3)
205. **PlotText** ( Indicators) – write text on the chart (AFL 2.80)
206. **PLOTVAPOVERLAY** (Exploration / Indicators) – plot Volume–At–Price overlay chart (AFL 2.4)
207. **PopupWindow** ( Miscellaneous functions) – display pop–up window (AFL 3.0)
208. **PREC** ( Math functions) – adjust number of decimal points of floating point number
209. **PREFS** ( Miscellaneous functions) – retrieve preferences settings (AFL 1.4)
210. **printf** ( String manipulation) – Print formatted output to the output window. (AFL 2.5)
211. **PVI** ( Indicators) – positive volume index
212. **RANDOM** ( Statistical functions) – random number (AFL 1.9)
213. **REF** ( Trading system toolbox) – reference past/future values of the array
214. **RELSTRENGTH** (Referencing other symbol data) – comparative relative strength (AFL 1.3)
215. **RequestTimedRefresh** ( Indicators) – forces periodical refresh of indicator pane (AFL 2.90)
216. **RestorePriceArrays** (Referencing other symbol data) – restore price arrays to original symbol (AFL 2.5)
217. **RMI** ( Indicators) – Relative Momentum Index (AFL 2.1)
218. **ROC** ( Indicators) – percentage rate of change
219. **ROUND** ( Math functions) – round number to nearest integer
220. **RSI** ( Indicators) – relative strength index
221. **RWI** ( Indicators) – random walk index
222. **RWIHI** ( Indicators) – random walk index of highs
223. **RWIL** ( Indicators) – random walk index of lows
224. **SAR** ( Indicators) – parabolic stop–and–reverse (AFL 1.3)
225. **Say** ( Miscellaneous functions) – speaks provided text (AFL 2.90)
226. **SECOND** ( Date/Time) – get current bar's second (AFL 2.0)
227. **SECTORID** ( Information / Categories) – get sector ID / name (AFL 1.8)
228. **SELECTEDVALUE** (Exploration / Indicators) – retrieves value of the array at currently selected date/time point (AFL 2.1)
229. **SetBacktestMode** ( Trading system toolbox) – Sets working mode of the backtester (AFL 3.0)
230. **SETBARSREQUIRED** ( Miscellaneous functions) – set number of previous and future bars needed for script/DLL to properly execute (AFL 2.1)
231. **SetChartBkColor** ( Indicators) – set background color of a chart (AFL 2.80)
232. **SetChartBkGradientFill** ( Indicators) – enables background gradient color fill in indicators (AFL 2.90)
233. **SetChartOptions** (Exploration / Indicators) – set/clear/overwrite defaults for chart pane options (AFL 2.70)
234. **SetCustomBacktestProc** ( Trading system toolbox) – define custom backtest procedure formula file (AFL 2.70)

235. **SetForeign** (Referencing other symbol data) – replace current price arrays with those of foreign security (AFL 2.5)
236. **SetFormulaName** ( Trading system toolbox) – set the name of the formula (AFL 2.5)
237. **SetOption** ( Trading system toolbox) – sets options in automatic analysis settings (AFL 2.3)
238. **SetPositionSize** ( Trading system toolbox) – set trade size (AFL 2.70)
239. **SetSortColumns** (Exploration / Indicators) – sets the columns which will be used for sorting in AA window (AFL 2.90)
240. **SETTRADEDELAYS** ( Trading system toolbox) – allows to control trade delays applied by the backtester (AFL 2.1)
241. **sign** ( Math functions) – returns the sign of the number/array (AFL 2.50)
242. **SIGNAL** ( Indicators) – macd signal line
243. **SIN** ( Math functions) – sine function
244. **sinh** ( Math functions) – hyperbolic sine function (AFL 2.80)
245. **SQRT** ( Math functions) – square root
246. **StaticVarGet** ( Miscellaneous functions) – gets the value of static variable (AFL 2.60)
247. **StaticVarGetText** ( Miscellaneous functions) – gets the value of static variable as string (AFL 2.60)
248. **StaticVarRemove** ( Miscellaneous functions) – remove static variable (AFL 2.80)
249. **StaticVarSet** ( Miscellaneous functions) – sets the value of static variable (AFL 2.60)
250. **StaticVarSetText** ( Miscellaneous functions) – Sets the value of static string variable. (AFL 2.60)
251. **STATUS** ( Miscellaneous functions) – get run–time AFL status information (AFL 1.65)
252. **STDERR** ( Statistical functions) – standard error (AFL 1.4)
253. **STDEV** ( Statistical functions) – standard deviation (AFL 1.4)
254. **STOCHD** ( Indicators) – stochastic slow %D
255. **STOCHK** ( Indicators) – stochastic slow %K
256. **StrExtract** ( String manipulation) – extracts given item (substring) from comma–separated string (AFL 2.4)
257. **StrFind** ( String manipulation) – find substring in a string (AFL 2.5)
258. **StrFormat** ( String manipulation) – Write formatted output to the string (AFL 2.5)
259. **StrLeft** ( String manipulation) – extracts the leftmost part (AFL 2.0)
260. **StrLen** ( String manipulation) – string length (AFL 1.5)
261. **StrMid** ( String manipulation) – extracts part of the string (AFL 2.0)
262. **StrReplace** ( String manipulation) – string replace (AFL 2.90)
263. **StrRight** ( String manipulation) – extracts the rightmost part of the string (AFL 2.0)
264. **StrToDateTime** ( String manipulation) – convert string to datetime (AFL 2.80)
265. **StrToLower** ( String manipulation) – convert to lowercase (AFL 2.80)
266. **StrToNum** ( String manipulation) – convert string to number (AFL 2.5)
267. **StrToUpper** ( String manipulation) – convert to uppercase (AFL 2.80)
268. **STUDY** ( Miscellaneous functions) – reference hand–drawn study (AFL 1.5)
269. **SUM** ( Moving averages, summation) – sum data over specified number of bars
270. **tan** ( Math functions) – tangent function (AFL 1.0)
271. **tanh** ( Math functions) – hyperbolic tangent function (AFL 2.80)
272. **TEMA** ( Moving averages, summation) – triple exponential moving average (AFL 2.0)
273. **TimeFrameCompress** (Time Frame functions) – compress single array to given time frame (AFL 2.5)
274. **TimeFrameExpand** (Time Frame functions) – expand time frame compressed array (AFL 2.5)
275. **TimeFrameGetPrice** (Time Frame functions) – retrieve O, H, L, C, V values from other time frame (AFL 2.5)
276. **TimeFrameMode** (Time Frame functions) – switch time frame compression mode (AFL 2.80)
277. **TimeFrameRestore** (Time Frame functions) – restores price arrays to original time frame (AFL 2.5)
278. **TimeFrameSet** (Time Frame functions) – switch price arrays to a different time frame (AFL 2.5)
279. **TIMENUM** ( Date/Time) – get current bar time (AFL 2.0)
280. **TRIN** ( Composites) – traders (Arms) index (AFL 1.2)
281. **TRIX** ( Indicators) – triple exponential smoothed price

- 282. **TROUGH** ( Basic price pattern detection) – trough (AFL 1.1)
- 283. **TROUGHBARS** ( Basic price pattern detection) – bars since trough (AFL 1.1)
- 284. **TSF** ( Statistical functions) – time series forecast (AFL 2.2)
- 285. **ULTIMATE** ( Indicators) – ultimate oscillator
- 286. **UNCISSUES** ( Composites) – unchanged issues (AFL 1.2)
- 287. **UNCVOLUME** ( Composites) – unchaged issues volume (AFL 1.2)
- 288. **VALUEWHEN** ( Trading system toolbox) – get value of the array when condition met (AFL 1.1)
- 289. **VarGet** ( Miscellaneous functions) – gets the value of dynamic variable (AFL 2.60)
- 290. **VarGetText** ( Miscellaneous functions) – gets the text value of dynamic variable (AFL 2.80)
- 291. **VarSet** ( Miscellaneous functions) – sets the value of dynamic variable (AFL 2.60)
- 292. **VarSetText** ( Miscellaneous functions) – sets dynamic variable of string type (AFL 2.80)
- 293. **VERSION** ( Miscellaneous functions) – get version info (AFL 1.9)
- 294. **WILDERS** ( Moving averages, summation) – Wilder's smoothing (AFL 1.4)
- 295. **WMA** ( Moving averages, summation) – weighted moving average (AFL 2.0)
- 296. **WRITEIF** (Exploration / Indicators) – commentary conditional text output
- 297. **WriteVal** (Exploration / Indicators) – write number or value of the array
- 298. **YEAR** ( Date/Time) – year (AFL 1.4)
- 299. **ZIG** ( Basic price pattern detection) – zig-zag indicator (AFL 1.1)
- 300. **\_DEFAULT\_NAME** (Exploration / Indicators) – retrive default name of the plot (AFL 2.70)
- 301. **\_N** (Exploration / Indicators) – no text output (AFL 2.1)
- 302. **\_PARAM\_VALUES** (Exploration / Indicators) – retrieve param values string (AFL 2.70)
- 303. **\_SECTION\_BEGIN** (Exploration / Indicators) – section begin marker (AFL 2.70)
- 304. **\_SECTION\_END** (Exploration / Indicators) – section end marker (AFL 2.70)
- 305. **\_SECTION\_NAME** (Exploration / Indicators) – retrieve current section name (AFL 2.70)
- 306. **\_TRACE** ( Miscellaneous functions) – print text to system debug viewer (AFL 2.4)

**#INCLUDE****Miscellaneous functions**  
(AFL 2.2)**– preprocessor include command****SYNTAX**     **#include****RETURNS**     nothing

**FUNCTION**     Includes external AFL files into your formula. Note 1: include statement need SINGLE backslashes in the path (this is quite the opposite to normal AFL sting parsing)  
 Note 2: using #include command may slow down formula execution even considering the fact that AmiBroker tries to include only once and cache pre-processed text  
 Note 3: that currently no error message is given if #include fails and this code is experimental.  
 Note 4: nesting #include commands is not supported  
 Note 5: by default files #included are cached by the AmiBroker. To turn off caching use #pragma nocache  
 before any #include statements. #include now accepts new way of specifying file names to include:

#include &lt;filename.afl&gt;

(note < > braces instead of " " ) if you specify the file name this way AmiBroker will look for the file in "standard include path" that is definable using new prefs setting in Tools->Preferences->AFL It makes much shorter to write includes and you can move include folder now without changing all AFL codes using #includes.

For example if you have set standard include path to "C:\AFL\MyIncludes" and write in your formula:

#include &lt;common.afl&gt;

AmiBroker will look for C:\AFL\MyIncludes\common.afl file

Also now #include reports file(s) not found in regular error message box. ?

**EXAMPLE**     #include "C:\Program Files\AmiBroker\AFL\common.afl"**SEE ALSO**     [#pragma\(\)](#) function**References:**

The **#include** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)

## #INCLUDE\_ONCE

– preprocessor include (once) command

Miscellaneous functions  
(AFL 2.70)

**SYNTAX**      *#include "formula file path"*

**RETURNS**     nothing

**FUNCTION**    Includes external AFL files into your formula. Similar to #include but #include\_once performs inclusion only once per formula. So if single formula has multiple #include\_once commands for the same file (for example because of drag-and-drop overlay) it prevents syntax errors that could occur due to repeated definitions of functions in included file. More information can be found in #include command docs.

**EXAMPLE**     #include\_once "myfile.afl"

**SEE ALSO**     [#include\(\)](#) function

### References:

The **#include\_once** function is used in the following formulas in AFL on–line library:

### More information:

[Updated on–line reference](#)

**#PRAGMA****– sets AFL pre-processor option****Miscellaneous functions**  
(AFL 2.4)**SYNTAX**     **#pragma optionname****RETURNS**     NOTHING

**FUNCTION**     Sets various AFL pre-processor options. Pre-processor is a part of AFL engine that processes formulas BEFORE they are executed. Currently the only task of pre-processor is to include external files via #include command.  
#pragma allows to change pre-processor behaviour.

Currently the only option available via #pragma is nocache

#pragma nocache  
causes that #included files are not cached so they are re-read with every execution

#pragma nocache  
must be placed before any #include commands. Note: between '#pragma' and 'nocache' there must be exactly SINGLE space

**Note 2: disabling caching may slow down execution of the formula (especially in indicators) !!!**

**EXAMPLE**     #pragma nocache  
                 #include "myfile.afl"

**SEE ALSO**     [#include\(\)](#) function

**References:**

The **#pragma** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)



**ABS****Math functions****– absolute value**

**SYNTAX**     *abs( NUMBER )*  
                  *abs( ARRAY )*

**RETURNS**    NUMBER  
                  ARRAY

**FUNCTION**    Calculates the absolute value of the NUMBER or ARRAY.

**EXAMPLE**     The formula "abs( -15 )" will return +15; the formula "abs( 15)" also returns +15.

**SEE ALSO****References:**

The **abs** function is used in the following formulas in AFL on–line library:

- [Absolute Breadth Index](#)
- [Adaptive Zones O/B &O/S Oscillator](#)
- [Adaptive Laguerre Filter, from John Ehlers](#)
- [ADXbuy](#)
- [Against all odds](#)
- [Analytic RSI formula](#)
- [Another Flb Level](#)
- [Application of Ehler filter](#)
- [AR\\_Prediction.afl](#)
- [Auto–Optimization Framework](#)
- [Bullish Percent Index 2 files combined](#)
- [Candle Identification](#)
- [Candle Pattern Function](#)
- [Candle Stick Analysis](#)
- [CandleStick Comentary--Help needed](#)
- [Candlestick Commentary](#)
- [Candlestick Commentary Modified](#)
- [Candlestick Commentary–modified](#)
- [CandleStochastics](#)
- [CCT Kaleidoscope](#)
- [Double top detection](#)
- [Ed Seykota's TSP: EMA Crossover System](#)
- [Elder Triple Screen Trading System.](#)
- [Head &Shoulders Pattern](#)
- [Hilbert Study](#)
- [Hurst Constant](#)
- [MA Difference 20 Period](#)
- [MACD and histogram divergence detection](#)
- [MACD commentary](#)
- [MACD indicator display](#)
- [Market Direction](#)
- [MultiCycle 1.0](#)
- [Multiple sinus noised](#)
- [Nonlinear Ehlers Filter](#)



- [nth \( 1 – 8 \) Order Polynomial Fit](#)
- [Option Calls, Puts and days till third friday.](#)
- [Pattern Recognition Exploration](#)
- [prakash](#)
- [Range Expansion Index](#)
- [Raw ADX](#)
- [RSI of Weekly Price Array](#)
- [SectorRSI](#)
- [The Mean RSIt](#)
- [The Mean RSIt \(variations\)](#)
- [tomy\\_frenchy](#)
- [Triangle exploration using PFChart](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [Tushar Chande's Projected Range](#)
- [TWS auto-export Executions-file parser](#)
- [Vertical Horizontal Filter](#)
- [Vertical Horizontal Filter \(VHF\)](#)
- [Vic Huebner](#)
- [Volatility Quality Index](#)

**More information:**

[Updated on-line reference](#)

## **ACCDIST** – accumulation/distribution

**Indicators****SYNTAX**     *AccDist()***RETURNS**     ARRAY**FUNCTION**     Calculates the Accumulation/ Distribution indicator.**EXAMPLE****SEE ALSO****References:**

The **AccDist** function is used in the following formulas in AFL on–line library:

- [accum/dist mov avg crossover SAR](#)
- [Bollinger band normalization](#)

**More information:**

[Updated on–line reference](#)

**ACOS****Math functions****– arccosine function****SYNTAX**     *acos( x )***RETURNS**    NUMBER, ARRAY**FUNCTION**    Returns the arccosine of NUMBER or ARRAY. The acos function returns the arccosine of x in the range 0 to pi radians. If x is less than –1 or greater than 1, acos returns an indefinite.**EXAMPLE****SEE ALSO**    [COS\(\)](#) function**References:**

The **acos** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**ADDCOLUMN****Exploration / Indicators****– add numeric exploration column**

(AFL 1.8)

**SYNTAX**     *AddColumn( array, name, format = 1.2, textColor = colorDefault, bkgndColor = colorDefault, width = -1 )*

**RETURNS**    NOTHING

**FUNCTION**    Adds a new column to the exploration result list. The column shows *array* values and has a caption of *name*. The values are formatted using *format* specification. By default all variables are displayed with 2 decimal digits, but you can change this by assigning a different value to this variable: 1.5 gives 5 decimal digits, 1.0 gives no decimal digits. (Note for advanced users: the integer part of this number can be used to pad formatted number with spaces – 6.0 will give no decimal digits but a number space-padded upto 6 characters.) Next two parameters allow to modify text and background color.

special format constants:

- **formatDateTime** – produces date time formatted according to your system settings

```
AddColumn( DateTime(), "Date / Time", formatDateTime );
```

- **formatChar** – allows outputting single ASCII character codes:

Example (produces signal file accepted by various other programs):

```
Buy=Cross(MACD(),Signal());
Sell=Cross(Signal(), MACD());
Filter=Buy OR Sell;
SetOption("NoDefaultColumns", True );
AddColumn( DateTime(), "Date", formatDateTime );
AddColumn( IIf( Buy, 66, 83 ), "Signal", formatChar );
```

- **width** parameter allows to control pixel width of the column

**EXAMPLE**     1. Simple column showing close price

```
addcolumn( Close, "Close price", 1.4 );
```

2. Colorful output

```
Filter =1;

AddColumn( Close, "Close", 1.2 );
AddColumn( MACD(), "MACD", 1.4 , IIf( MACD() > 0, colorGreen,
colorRed ) );
AddTextColumn( FullName(), "Full name", 77 , colorDefault, IIf(
Close < 10, colorLightBlue, colorDefault ) );
```

**SEE ALSO**     [ADDTXTCOLUMN\(\)](#) function

**References:**

The **AddColumn** function is used in the following formulas in AFL on–line library:

- 'DE\*H37href='http://www.amibroker.com/library/detail.php?id=718' target='\_blank'>abosliman2005m

- ADXbuy
- AFL Example
- AFL Example – Enhanced
- Alert Output As Quick Rewiev
- Aroon Indicators
- Auto–Optimization Framework
- Black Scholes Option Pricing
- Bottom Trader
- Bull Fear / Bear Fear
- Bullish Percent Index 2 files combined
- Calculate composites for tickers in list files
- CAMSLIM Cup and Handle Pattern AFL
- Count Tickers in Watchlist
- Ed Seykota's TSP: EMA Crossover System
- Ed Seykota's TSP: Support and Resistance
- ekeko price chart
- Elder Impulse Indicator V2
- End Of Year Trading
- Follow the Leader
- Gordon Rose
- half–automated Trading System
- MACD and histogram divergence detection
- Monthly bar chart
- Negative ROC Exporation
- nth ( 1 – 8 ) Order Polynomial Fit
- pattern correlation
- Positive ROC Exploration
- Price Persistency
- Ranking and sorting stocks
- Relative Strength
- RSI "based" Trading System
- RSI Double–Bottom
- RSI Trendlines and Wedges
- RUTVOL timing signal with BB Scoring routine
- Sainath Sidgiddi
- SAR–ForNextBarStop
- SectorRSI
- Simple Candle Exploration
- Smoothed RSI Buy Signals
- STD\_STK Multi
- StochD\_StochK Single.afl
- Stops Implementation in AFS
- Strength and Weakness
- TAZ Trading Method Exploration
- Trend Detection
- Trend exploration with multiple timeframes
- Triangle exploration using PFChart
- Triangular Moving Average new
- Using From and To dates from Auto Analysis in Code
- Volume – compared with Moving Avg (100%)
- Weekly chart
- Weighted Index

- [Williams %R Exploration](#)

**More information:**

[Updated on-line reference](#)

**ADDEXTCOLUMN****Exploration / Indicators****– add text exploration column**

(AFL 1.8)

**SYNTAX**     *AddTextColumn( string, name, format = 1.2, textColor = colorDefault, bkgndColor = colorDefault, width = -1 )*

**RETURNS**    NOTHING

**FUNCTION**   Adds a new text column to the exploration result list. The column shows *text* and has a caption of *name*.  
Next two parameters allow to modify text and background color.  
Width parameter allows to control pixel width of the column

**EXAMPLE**     addtextcolumn( GroupID( 1 ), "The name of the group");

**SEE ALSO**     [ADDCOLUMN\(\)](#) function

**Comments:**

<b>Tomasz Janeczko</b>  2005-08-10 06:35:35	Please note that AddTextColumn takes single string as a parameter, so you can only display text that does NOT vary on bar-by-bar basis.
--	---

**References:**

The **AddTextColumn** function is used in the following formulas in AFL on-line library:

- [AFL Example](#)
- [AFL Example – Enhanced](#)
- [Alert Output As Quick Rewiev](#)
- [Bottom Trader](#)
- [Dave Landry PullBack Scan](#)
- [Elder Impulse Indicator V2](#)
- [Elder Triple Screen Trading System.](#)
- [MACD and histogram divergence detection](#)
- [pattern correlation](#)
- [Relative Strength](#)
- [Trend exploration with multiple timeframes](#)
- [Triangular Moving Average new](#)
- [WLBuidProcess](#)

**More information:**

[Updated on-line reference](#)

**ADDTOCOMPOSITE****Composites**  
(AFL 2.0)**– add value to composite ticker****SYNTAX**     **AddToComposite( array, "ticker", "field", flags = atcFlagDefaults )****RETURNS**    NOTHING**FUNCTION**    Allows you to create composite indicators with ease. [More info...](#)

Parameters:

array – the array of values to be added to "field" in "ticker" composite symbol

"ticker" – the ticker of composite symbol. It is advised to use ~comp (tilde at the beginning)

newly added composites are assigned to group 253 by default and

have "use only local database" feature switched on for proper operation with external

sources possible field codes: "C" – close, "O" – open, "H" – high, "L" – low, "V" – volume, "I"

– open interest, "X" – updates all OHLC fields at once

flags – contains the sum of following values

- atcFlagDeleteValues = 1 – deletes all previous data from composite symbol at the beginning of scan (recommended)
- atcFlagCompositeGroup = 2 – put composite ticker into group 253 and EXCLUDE all other tickers from group 253 (avoids adding composite to composite)
- atcFlagTimeStamp = 4 – put last scan date/time stamp into FullName field
- atcFlagEnableInBacktest = 8 – allow running AddToComposite in backtest/optimization mode
- atcFlagEnableInExplore = 16 – allow running AddToComposite in exploration mode
- atcFlagResetValues = 32 – reset values at the beginning of scan (not required if you use atcFlagDeleteValues)
- atcFlagDefaults = 7  
(this is a composition of atcFlagResetValues | atcFlagCompositeGroup | atcFlagTimeStamp flags)
- atcFlagEnableInPortfolio = 64 – allow running AddToComposite in custom portfolio backtester phase
- atcFlagEnableInIndicator = 128 – allow running AddToComposite in indicator mode

AddToComposite function also detects the context in which it is run

(it works ONLY in scan mode, unless atcFlagEnableInBacktest or atcFlagEnableInExplore flags are specified) and does NOT affect composite ticker when run in Indicator or Commentary mode, so it is now allowed to join scan and indicator into single formula.

**EXAMPLE**     `AddToComposite( MACD() > 0, "~BullMACD", "V");`  
                   `Graph0 = Foreign( "~BullMACD", "V" );`  
                   `// Now you can use the same formula in scan AND indicator`

**SEE ALSO****References:**The **AddToComposite** function is used in the following formulas in AFL on–line library:

- [30 Week Hi Indicator – Calculate](#)
- [52 Week New High–New Low Index](#)



- [Bullish Percent Index 2 files combined](#)
- [Bullish Percent Index 2004](#)
- [Calculate composites for tickers in list files](#)
- [Compare Sectors against Tickers](#)
- [Index of 30 Wk Highs Vs Lows](#)
- [Market Direction](#)
- [Overbought issues, Oversold issues](#)
- [RUTVOL timing signal with BB Scoring routine](#)
- [SectorRSI](#)
- [Stochastic Divergences, PDI, NDI](#)
- [Stochastic OSI & OBI](#)
- [The Mean RSI](#)
- [The Mean RSI \(variations\)](#)
- [The Relative Slope Pivots](#)
- [Trending or Trading ?](#)
- [Weighted Index](#)
- [WLBuildProcess](#)

**More information:**

[Updated on-line reference](#)

**ADLINE**  
**– advance/decline line****Composites**  
(AFL 1.2)**SYNTAX**     **ADLine()****RETURNS**     ARRAY**FUNCTION**     Calculates Advance/Dcline line indicator**EXAMPLE**     adline()**SEE ALSO****References:**

The **ADLine** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**ADVISSUES****Composites****– advancing issues**

(AFL 1.2)

**SYNTAX**     *AdvIssues()***RETURNS**     ARRAY**FUNCTION**     Returns the number of advancing issues for a given market (the one that currently analysed stock belongs to)**EXAMPLE**     *advissues()***SEE ALSO****References:**

The **AdvIssues** function is used in the following formulas in AFL on–line library:

- [Absolute Breadth Index](#)
- [Breadth Thrust](#)
- [McClellan Oscillator](#)
- [McClellan Summation Index](#)

**More information:**

[Updated on–line reference](#)

**ADVOLUME**  
**– advancing issues volume****Composites**  
(AFL 1.2)**SYNTAX**     *AdvVolume()***RETURNS**    ARRAY**FUNCTION**    Returns the volume of advancing issues for a given market (the one that currently analysed stock belongs to)**EXAMPLE**     *advvolume()***SEE ALSO****References:**

The **AdvVolume** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

## **ADX** – average directional movement index

**Indicators**  
(AFL 1.3)

**SYNTAX**     *adx( period = 14 )*

**RETURNS**     ARRAY

**FUNCTION**     Calculates Average Directional Index indicator

**EXAMPLE**     *adx(), adx(20)*

### **SEE ALSO**

#### **References:**

The **ADX** function is used in the following formulas in AFL on–line library:

- [ADX Indicator – Colored](#)
- [ADXR](#)
- [Bollinger band normalization](#)
- [Bull Fear / Bear Fear](#)
- [Dave Landry Pullbacks](#)
- [DMI Spread Index](#)
- [ekeko price chart](#)
- [Gordon Rose](#)
- [Mndahoo ADX](#)
- [TAZ Trading Method Exploration](#)

#### **More information:**

[Updated on–line reference](#)

## ALERTIF

### – trigger alerts

Trading system toolbox  
(AFL 2.1)

**SYNTAX**     *AlertIf( BOOLEAN\_EXPRESSION, command, text, type = 0, flags = 1+2+4+8, lookback = 1 );*

**RETURNS**     nothing

**FUNCTION**    Triggers alert action if BOOLEAN\_EXPRESSION is true.

1. *BOOLEAN\_EXPRESSION* is the expression that if evaluates to True (non zero value) triggers the alert. If it evaluates to False (zero value) no alert is triggered. Please note that only *lookback* most recent bars are considered.

2. The *command* string defines the action taken when alert is triggered. If it is empty the alert *text* is simply displayed in the Alert output window (View->Alert Output). Other supported values of *command* string are:

SOUND *the-path-to-the-WAV-file*

EMAIL

EXEC *the-path-to-the-file-or-URL*

SOUND command plays the WAV file once.

EMAIL command sends the e-mail to the account defined in the settings (Tools->Preferences->E-mail). The format of the e-mail is as follows: Subject: Alert type\_name (*type*) Ticker on Date/Time

Body: *text*

EXEC command launches external application or file or URL specified after EXEC command. are attached after file name and *text* is attached at the end

3. *Text* defines the text that will be printed in the output window or sent via e-mail or added as argument to the application specified by EXEC command

4. *Type* defines type of the alert. Pre-defined types are 0 – default, 1 – buy, 2 – sell, 3 – short, 4 – cover. YOU may specify higher values and they will get name "other"

5. *Flags* control behaviour of AlertIF function. This field is a combination (sum) of the following values:

( 1 – display text in the output window, 2 – make a beep (via computer speaker), 4 – don't display repeated alerts having the same type, 8 – don't display repeated alerts having the same date/time) By default all these options are turned ON.

6. *lookback* parameter controls how many recent bars are checked

**EXAMPLE**

```
Buy = Cross( MACD(), Signal() );
Sell = Cross( Signal(), MACD() );
Short = Sell;
Cover = Buy;
AlertIF( Buy, "EMAIL", "A sample alert on "+FullName(), 1 );
AlertIF( Sell, "SOUND C:\\Windows\\Media\\Ding.wav", "Audio alert",
2 );
AlertIF( Short, "EXEC Calc.exe", "Launching external application", 3
```

```
);  
AlertIF( Cover, "", "Simple text alert", 4 );
```

Note EXEC command uses ShellExecute function and allows not only EXE files but URLs too.

**SEE ALSO****References:**

The **AlertIf** function is used in the following formulas in AFL on-line library:

- [AFL Example – Enhanced](#)
- [Alert Output As Quick Rewiev](#)
- [ATR Trading System](#)
- [CCI\(20\) Divergence Indicator](#)
- [Stock price AlertIf](#)
- [Trading ATR 10–1](#)

**More information:**

[Updated on-line reference](#)

**ALMOSTEQUAL****Math functions****– rounding error insensitive comparison**

(AFL 2.80)

**SYNTAX**     *AlmostEqual( x, y, ulps = 5 )***RETURNS**    NUMBER  
              ARRAY**FUNCTION**    This is a helper function for comparing floating point numbers. It returns True if x and y are equal or almost equal upto defined accuracy (ulps). It is recommended to use this function instead of equality check (==) as it leads to more reliable comparisons and less headache caused by IEEE floating point accuracy issues.

Parameters:

- **x, y** – the numbers or arrays to be compared,
- **ulps** stands for "units in last place" and represents maximum relative error of the comparison. Since 32 bit IEEE floating point numbers have accuracy of 7 significant digits, 1 unit in last place(ulp) represents relative error of 0.00001 %. The default value of ulps parameter is 5 which gives roughly 0.00005% "comparison sensitivity".

Thanks to Bruce Dawson for his fast routine.

**EXAMPLE**    `if( 1/3 == 0.3333333 )`  
               `{`  
                   `printf( "32-bit Floating point IEEE exact equality\n" );`  
               `}`

`if( AlmostEqual( 1/3, 0.3333333 ) )`  
               `{`  
                   `printf( "Numbers are almost equal\n" );`  
               `}`

**SEE ALSO****References:**The **AlmostEqual** function is used in the following formulas in AFL on–line library:**More information:**[Updated on–line reference](#)



**AMA****Moving averages, summation****– adaptive moving average**

(AFL 1.5)

**SYNTAX**     **ama( ARRAY, SMOOTHINGFACTOR )****RETURNS**     ARRAY**FUNCTION**     calculates adaptive moving average – similar to EMA() but smoothing factor could be time-variant (array).

**EXAMPLE**     The example of volatility-weighted adaptive moving average formula: graph0 = ema( close, 15 );  
                   fast = 2/(2+1);  
                   slow = 2/(30+1);  
                   dir=abs(close-ref(close,-10));  
                   vol=sum(abs(close-ref(close,-1)),10);  
                   ER=dir/vol;  
                   sc =( ER\*(fast-slow)+slow)^2; graph0 = **ama**( close, sc );

**SEE ALSO****Comments:**

<b>Tomasz Janeczko</b>	output = AMA( input, factor )
2006-04-26 20:13:15	is equivalent to the following looping code:  <pre>for( i = 1; i &lt; BarCount; i++ ) {     output[ i ] = factor[ i ] * input[ i ] + ( 1 - factor[ i ] ) * output[ i - 1 ]; }</pre>

**References:**

The **AMA** function is used in the following formulas in AFL on-line library:

- [Application of Ehler filter](#)
- [Auto-Optimization Framework](#)
- [Hilbert Study](#)
- [IFT of RSI – Multiple TimeFrames](#)
- [INTRADAY HEIKIN ASHI new](#)
- [Pivots for Intraday Forex Charts](#)

**More information:**

[Updated on-line reference](#)

**AMA2****Moving averages, summation****– adaptive moving average**

(AFL 1.5)

**SYNTAX**     *ama2( ARRAY, SMOOTHINGFACTOR, FEEDBACKFACTOR )***RETURNS**     ARRAY

**FUNCTION**     calculates adaptive moving average – similar to EMA() but smoothing factor could be time-variant (array).  
 AMA2 has a separate control of feedbackfactor which is normally (1-SMOOTHINGFACTOR). Internally this function works like this: today\_ama = SMOOTHINGFACTOR \* array + FEEDBACKFACTOR \* yesterday\_ama

**EXAMPLE**     The example of volatility-weighted adaptive moving average formula: graph0 = ema( close, 15 );  
                   fast = 2/(2+1);  
                   slow = 2/(30+1);  
                   dir=abs(close-ref(close,-10));  
                   vol=sum(abs(close-ref(close,-1)),10);  
                   ER=dir/vol;  
                   sc =( ER\*(fast-slow)+slow)^2; graph0 = **ama2**( close, sc, 1-sc);

**SEE ALSO****References:**

The **AMA2** function is used in the following formulas in AFL on-line library:

- [Candle Stick Analysis](#)

**More information:**

[Updated on-line reference](#)

**APPLYSTOP**

Trading system toolbox

**– apply built-in stop**

(AFL 1.7)

**SYNTAX**     *ApplyStop( type, mode, amount, exitatstop, volatile = False, ReEntryDelay = 0 )***RETURNS**     nothing**FUNCTION**     controls built-in stops from the formula level (allows optimization of stops)

Parameters:

*type* =

0 = stopTypeLoss – maximum loss stop,

1 = stopTypeProfit – profit target stop,

2 = stopTypeTrailing – trailing stop,

3 = stopTypeNBar – N-bar stop

*mode* =

0 – disable stop (stopModeDisable),

1 – amount in percent (stopModePercent), or number of bars for N-bar stop (stopModeBars),

2 – amount in points (stopModePoint);

3 – amount in percent of profit (risk)

*amount* =

percent/point loss/profit trigger/risk amount.

This could be a number (static stop level) or an array (dynamic stop level)

*ExitAtStop*

ExitAtStop = 0 – means check stops using only trade price and exit at regular trade price (if you are trading on close it means that only close price will be checked for exits and exit will be done at close price)

ExitAtStop = 1 – check High–Low prices and exit intraday on price equal to stop level on the same bar when stop was triggered

ExitAtStop = 2 – check High–Low prices but exit NEXT BAR on regular trade price.

*volatile* –

decides if amount (or distance) (3rd parameter) is sampled at the trade entry and remains fixed during the trade (Volatile = FALSE – old behaviour) or if can vary during the trade (Volatile = TRUE) (allows single line Chandelier exit implementation)

*ReEntryDelay* –

how many bars to wait till entering the same stock is allowed.

**Note on using stops:****Scenario 1:**

you trade on next bar OPEN and want to exit intraday on stop price

Correct settings:

ActivateStopsImmediately turned ON

ExitAtStop = 1  
 Trade delays set to one  
 Trade price set to open

**Scenario 2:**

you trade on today's close and want to exit intraday on stop price

Correct settings:

ActivateStopsImmediately turned OFF

ExitAtStop = 1

Trade delays set to zero

Trade price set to close

**Scenario 3:**

you trade on next day OPEN and want to exit by stop on OPEN price when PREVIOUS day H-L range hits stop

Correct settings:

ExitAtStop = 2 (NEW)

Trade delays set to one

Trade price set to open

- a) (if you want to have stops executed AFTER regular signals, so cash from stopped out positions is NOT available to enter trades the same day)

**ActivateStopsImmediately turned ON**

- b) (if you want to have stops executed BEFORE regular signals, so cash from stopped out positions IS available to enter new trades the same day)

**ActivateStopsImmediately turned OFF**

**Scenario 4:**

you trade on today's close and want to exit only when today's close price hits the stop level

Correct settings:

ActivateStopsImmediately turned OFF

ExitAtStop = 0

Trade delays set to zero

Trade price set to close

```
EXAMPLE  /* max loss stop optimization */

ApplyStop(stopTypeLoss,
           stopModePercent,
           Optimize( "max. loss stop level", 10, 2, 30, 1 ),
           True );

/* single-line implementation of Chandelier exit */

ApplyStop(stopTypeTrailing, stopModePoint, 3*ATR(14), True, True );
```

```
/* N-bar stop */
ApplyStop( stopTypeNBar, stopModeBars, 5 );
```

**SEE ALSO****Comments:**

<b>Herman van den Bergen</b> psytek@magma.ca 2003-02-23 09:53:51	If you are trading at the Close with zero delays be sure to unmark "Activate Stops Immediately" in Settings.
<b>Corey Saxe</b> csaxe@nwi.net 2003-03-01 23:33:13	For visual conformation of ApplyStop function, add the following lines below your ApplyStop formula in Indicator Builder:  Equity(1); // THIS EVALUATES STOPS  Plot(Sell==4,"ApplyStop Sell",colorRed,1 styleOwnScale); Plot(Cover==4,"ApplyStop Cover",colorGreen,1 styleOwnScale);
<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2004-08-28 03:14:12	If two or more different stops are triggered on the VERY SAME bar then they are evaluated in this fixed order:  Fixed Ruin stop (loosing 99.96% of the starting capital) Max. loss stop Profit target stop Trailing stop N-bar stop
<b>Graham Kavanagh</b> gkavanagh@e-wire.net.au 2004-09-30 21:55:42	from equity comments  Depending on kind of the stop various values are written back to sell/cover array to enable you to distinguish if given signal was generated by regular rule or by stop.  1 – regular exit 2 – max. loss 3 – profit target 4 – trailing 5 – n-bar stop 6 – ruin stop
<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2005-03-01 17:10:39	ExitAtStop has a new meaning for N-BAR stop type. If ExitAtStop = 0 then N-bar stop has the lowest priority (so if for example profit target stop is hit on the same bar then profit target is evaluated first) If ExitAtStop = 1 then N-bar stop has highest priority and it is evaluated before all other stops. The same effect is obtained by checking "Has priority" box in AA Settings window.
<b>Tomasz Janeczko</b> tj-/nospam/@amibroker.com 2006-01-13 11:41:32	ApplyStop function is designed to be used to simulate stop orders placed at the exchange or simulated by the brokerage  Please read this how such stops operate: <a href="http://www.interactivebrokers.com/en/trading/orders/stop.php?ib_entity=uk">http://www.interactivebrokers.com/en/trading/orders/stop.php?ib_entity=uk</a> <a href="http://www.interactivebrokers.com/en/trading/orders/trailingStops.php?ib_entity=uk">http://www.interactivebrokers.com/en/trading/orders/trailingStops.php?ib_entity=uk</a>

**References:**

The **ApplyStop** function is used in the following formulas in AFL on–line library:

- [ATR Study](#)
- [danningham penetration](#)
- [Ed Seykota's TSP: EMA Crossover System](#)
- [Follow the Leader](#)
- [RUTVOL timing signal with BB Scoring routine](#)
- [SectorRSI](#)
- [The D\\_oscillator](#)
- [The Three Day Reversal](#)
- [Trend Continuation Factor](#)

**More information:**

[Updated on–line reference](#)

**ASC****String manipulation****– get ASCII code of character**

(AFL 2.80)

**SYNTAX**     *Asc( string, pos = 0 )***RETURNS**    NUMBER

**FUNCTION**    Returns the ANSI character code corresponding to the first letter in a string (if position is not specified) or code of character at specified position. If you don't specify position (pos argument) then first character is used. Negative values of pos reference characters counting from the end of string.

Useful for creation of exploration that displays single letters for signals instead of numbers.

**EXAMPLE**     `Buy = Cross(MACD(),Signal());`  
                  `Sell = Cross(Signal(),MACD());`

`Filter = Buy OR Sell;`

`AddColumn( IIf( Buy, Asc("B"), Asc("S")), "Signal", formatChar );`

**SEE ALSO**    `AddColumn()` function**References:**

The **Asc** function is used in the following formulas in AFL on–line library:

- [Ed Seykota's TSP: Support and Resistance](#)

**More information:**

[Updated on–line reference](#)

**ASIN****Math functions****– arcsine function****SYNTAX**     *asin( x )***RETURNS**    NUMBER, ARRAY**FUNCTION**   Returns the arcsine of NUMBER or ARRAY. The asin function returns the arcsine of x in the range  $-\pi/2$  to  $\pi/2$  radians. If x is less than  $-1$  or greater than  $1$ , asin returns an indefinite**EXAMPLE****SEE ALSO**    [SIN\(\)](#) function**References:**

The **asin** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)



**ATAN****Math functions****– arc tan**

**SYNTAX**     *atan( NUMBER ),*  
                 *atan( ARRAY )*

**RETURNS**    NUMBER  
                 ARRAY

**FUNCTION**   Returns the arc tangent of NUMBER or ARRAY. The value is returned in radians

**EXAMPLE**    The formula "atan( 1.00 )" returns PI/4

**SEE ALSO****References:**

The **atan** function is used in the following formulas in AFL on–line library:

- [Andrews Pitchfork](#)
- [Andrews PitchforkV3.3](#)
- [AR\\_Prediction.afl](#)
- [DMI Spread Index](#)
- [Hilbert Study](#)
- [Moving Average "Crash" Test](#)
- [Multiple sinus noised](#)
- [Schiff Lines](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**

[Updated on–line reference](#)

**ATAN2****Math functions****– calculates arctangent of y/x**

(AFL 2.90)

**SYNTAX**     *atan2( y, x )***RETURNS**    NUMBER or ARRAY

**FUNCTION**    atan2 returns the arctangent of y/x. If x is 0, atan2 returns 0. If both parameters of atan2 are 0, the function returns 0. atan2 returns a value in the range –PI to +PI radians, using the signs of both parameters to determine the quadrant of the return value.

**EXAMPLE**     `ffc = FFT(data,Len);  
for( i = 0; i < Len - 1; i = i + 2 )  
{  
    amp[ i ] = amp[ i + 1 ] = sqrt(ffc[ i ]^ 2 + ffc[ i + 1 ]^2);  
    phase[ i ] = phase[ i + 1 ] = atan2( ffc[ i + 1 ], ffc[ i ] );  
}`

**SEE ALSO**     *atan()* function

**References:**

The **atan2** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**ATR****– average true range****Indicators**  
(AFL 1.3)**SYNTAX**     *atr( period )***RETURNS**     ARRAY**FUNCTION**     Calculates Average True Range indicator**EXAMPLE**     atr(7)**SEE ALSO****Comments:**

<b>Bob Jagow</b> bjagow@charterr.net 2003-02-06 23:37:50	For other MAs, use ATR(1) to get the True Range.  E.g., MA(ATR(1),period), WMA(ATR(1),period), etc.
<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2005-02-03 06:46:39	Note that original formulation of ATR (the one that AmiBroker implements) uses WILDERS smoothing (not simple moving average)  For more details check this page: <a href="http://stockcharts.com/education/IndicatorAnalysis/indic_ATR.html">http://stockcharts.com/education/IndicatorAnalysis/indic_ATR.html</a>

**References:**

The **ATR** function is used in the following formulas in AFL on-line library:

- [AR\\_Prediction.afl](#)
- [ATR Study](#)
- [ATR Trading System](#)
- [BB squeeze](#)
- [Bollinger – Keltner Bands](#)
- [Bollinger Band Gap](#)
- [Bow tie](#)
- [Chandelier Exit](#)
- [Chandelier Exit or Advanced Trailing Stop](#)
- [Chandelier Plugin](#)
- [Gartley 222 Pattern Indicator](#)
- [Gordon Rose](#)
- [Keltner Channel](#)
- [Peterson](#)
- [PF Chart – Close – April 2004](#)
- [Position Sizing and Risk Price Graph](#)
- [Position Sizing and Risk Price Graph – 2](#)
- [Raw ADX](#)
- [Renko Chart](#)
- [SectorRSI](#)
- [SF Entry,Stop, PT Indicator](#)
- [STARC Bands](#)
- [Steve Woods' Cum. Vol. Float + Cum. Vol. Channels](#)
- [STO &MACD Buy Signals with Money-Management](#)
- [Trading ATR 10-1](#)

- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [Volatility Quality Index](#)
- [Volatility System](#)
- [Weinberg's The Range Indicator](#)

**More information:**

[Updated on-line reference](#)

**BARINDEX****Date/Time**  
(AFL 2.3)**– get zero–based bar number****SYNTAX**     *BarIndex()***RETURNS**     ARRAY**FUNCTION**     returns zero–based bar number – the same as Cum(1)–1 but it is much faster than Cum(1) when used in Indicators**EXAMPLE**     `ThisIsLastBar = BarIndex() == LastValue( BarIndex() );`**SEE ALSO**     [CUM\(\)](#) function**Comments:**

<b>Tomasz Janeczko</b> tj ---at--- amibroker.com 2004–07–23 07:07:29	When QuickAFL is ON, the BarIndex() may not be equal with array item index.  Actual array item corresponding to bar index can be found this way:  <pre>bi = BarIndex(); arrayitem = SelectedValue( bi ) – bi[ 0 ]; "Close at selected bar:" + Close[ arrayitem ];</pre>
--	---

**References:**

The **BarIndex** function is used in the following formulas in AFL on–line library:

- [Adaptive Price Channel](#)
- [Andrews Pitchfork](#)
- [Andrews PitchforkV3.3](#)
- [Another Flb Level](#)
- [AR\\_Prediction.afl](#)
- [Auto–Optimization Framework](#)
- [Bullish Percent Index 2004](#)
- [Candle Identification](#)
- [Candle Stick Analysis](#)
- [Candle Stick Demo](#)
- [CCI\(20\) Divergence Indicator](#)
- [Chandelier Exit](#)
- [crMathLib](#)
- [Cycle Highlighter](#)
- [Cycle Highlighter \(auto best–fit\)](#)
- [Date\\_To\\_Num\(\), Time\\_To\\_Num\(\)](#)
- [Elder safe Zone Long + short](#)
- [FirstBarIndex\(\), LastBarIndex\(\)](#)
- [Gordon Rose](#)
- [Hurst "Like" DE](#)
- [Linear Regression Line w/ Std Deviation Channels](#)
- [MACD indicator display](#)
- [Multiple sinus noised](#)
- [nth \( 1 – 8 \) Order Polynomial Fit](#)

- [pattern correlation](#)
- [PF Chart – Close – April 2004](#)
- [Pivot Finder](#)
- [Rea Time Daily Price Levels](#)
- [Relative Strength Multichart of up to 10 tickers](#)
- [Schiff Lines](#)
- [tomy\\_frenchy](#)
- [Trend Detection](#)
- [Trend exploration with multiple timeframes](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [Using From and To dates from Auto Analysis in Code](#)
- [WLBuildProcess](#)
- [Zig Zag Indicator with Valid Entry and Exit Points](#)

**More information:**

[Updated on-line reference](#)

**BARSSINCE**

Trading system toolbox

**– bars since****SYNTAX**      *BarsSince( ARRAY )***RETURNS**    ARRAY**FUNCTION**    Calculates the number of bars (time periods) that have passed since ARRAY was true (or 1)**EXAMPLE**     barssince( macd() < 0 )**SEE ALSO****References:**

The **BarsSince** function is used in the following formulas in AFL on–line library:

- [Andrews PitchforkV3.3](#)
- [AR\\_Prediction.afl](#)
- [Auto–Optimization Framework](#)
- [CCI Woodies Style](#)
- [CCI\(20\) Divergence Indicator](#)
- [Cole](#)
- [Commodity Channel Index](#)
- [Cycle Highlighter](#)
- [Cycle Highlighter \(auto best–fit\)](#)
- [Divergences](#)
- [ekeko price chart](#)
- [Elder Impulse Indicator V2](#)
- [FastStochK FullStochK–D](#)
- [Follow the Leader](#)
- [Fund Screener](#)
- [Gann Swing Chart](#)
- [IntraDay Open Marker](#)
- [Lagging MA–Xover](#)
- [MACD and histogram divergence detection](#)
- [MACD commentary](#)
- [MACD indicator display](#)
- [Market Profile &Market Volume Profile](#)
- [Moving Averages NoX](#)
- [Performance Check](#)
- [Peterson](#)
- [prakash](#)
- [Price Persistency](#)
- [Rea Time Daily Price Levels](#)
- [Relative Strength Index](#)
- [Schiff Lines](#)
- [Stochastic Divergence, negative](#)
- [Stochastic Divergence, positive](#)
- [Stochastic Divergences, PDI, NDI](#)
- [Stochastic Fast%K and Full](#)
- [Stops Implementation in AFS](#)
- [TD sequential](#)

- [The Fibonacci behavior](#)
- [The Three Day Reversal](#)
- [Trend Analysis\\_Comentary](#)
- [Triangular Moving Average](#)
- [Triangular Moving Average new](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [Williams %R with 9 period signal line](#)
- [Williams Alligator system](#)
- [Zig Zag Indicator with Valid Entry and Exit Points](#)

**More information:**

[Updated on-line reference](#)



**BBANDBOT****Indicators****– bottom bollinger band****SYNTAX**     *BBandBot( ARRAY, periods = 15, width = 2 )***RETURNS**     ARRAY**FUNCTION**     Calculates the bottom Bollinger Band of ARRAY shifted down *width* standard deviations (using *periods* averaging range ).**EXAMPLE**     bbandbot( close, 10, 2 )**SEE ALSO****References:**

The **BBandBot** function is used in the following formulas in AFL on–line library:

- [Bollinger – Keltner Bands](#)
- [Bollinger band normalization](#)
- [Bottom Trader](#)
- [ekeko price chart](#)
- [INTRADAY HEIKIN ASHI new](#)
- [prakash](#)
- [RUTVOL timing signal with BB Scoring routine](#)
- [Volatility Breakout with Bollinger Bands](#)

**More information:**

[Updated on–line reference](#)

**BBANDTOP****Indicators****– top bollinger band****SYNTAX**     *BBandTop( ARRAY, periods = 15, width = 2 )***RETURNS**     ARRAY**FUNCTION**     Calculates the bottom Bollinger Band of ARRAY shifted up *width* standard deviations (using *periods* averaging range ).**EXAMPLE**     `bbandtop( close, 10, 2 )`**SEE ALSO****References:**

The **BBandTop** function is used in the following formulas in AFL on–line library:

- [Bollinger – Keltner Bands](#)
- [Bollinger Band Gap](#)
- [Bollinger band normalization](#)
- [ekeko price chart](#)
- [INTRADAY HEIKIN ASHI new](#)
- [prakash](#)
- [RUTVOL timing signal with BB Scoring routine](#)
- [Volatility Breakout with Bollinger Bands](#)

**More information:**

[Updated on–line reference](#)

**BEGINVALUE****Date/Time**  
(AFL 2.3)**– Value of the array at the begin of the range****SYNTAX**     *BeginValue( ARRAY )***RETURNS**     NUMBER**FUNCTION**     This function gives the single value (number) of the ARRAY at the beginning of the selected range. If no range is marked then the value at the first bar is returned.

To select the range you have to double click in the chart at the beginning of the range and then double click in the chart at the end of the range. Then > < markers will appear above date axis.

**EXAMPLE**     1. Simple commentary:

```
WriteVal( BeginValue( DateTime() ), formatDateTime );
WriteVal( EndValue( DateTime() ), formatDateTime );
"Percentage change of close is " +
WriteVal( 100 * (EndValue( Close ) - BeginValue( Close
))/BeginValue( Close ) ) + "%";
```

2. Get the number of bars in the range and calculate some stats for that range:

```
Period = EndValue( BarIndex() ) - BeginValue( BarIndex() );
StandardDeviationInTheRange = EndValue( StDev( Close, Period ) );
```

**SEE ALSO**     [ENDVALUE\(\)](#) function**References:**

The **BeginValue** function is used in the following formulas in AFL on–line library:

- [Adaptive Price Channel](#)
- [Another Fib Level](#)
- [Elder safe Zone Long + short](#)
- [Futures – Dollar Move Indicator](#)
- [nth \( 1 – 8 \) Order Polynomial Fit](#)
- [pattern correlation](#)
- [Relative Strength Multichart of up to 10 tickers](#)

**More information:**

[Updated on–line reference](#)

**CATEGORYADDSYMBOL****Information / Categories****– adds a symbol to a category**

(AFL 2.5)

**SYNTAX**     *CategoryAddSymbol( symbol, category, number )***RETURNS**    NOTHING

**FUNCTION**    The **CategoryAddSymbol** function adds the *symbol* to given category, note that for markets, groups, industries 'adding' means moving from one category to another, since the symbol is assigned always to one and only one market, group, industry and sector. This limitation does not apply to watchlists, favorites, and index categories. When *symbol* string is empty ("") then current symbol is used.

*category* is one of the following:

- categoryMarket
- categoryGroup
- categorySector
- categoryIndustry
- categoryWatchlist
- categoryFavorite
- categoryIndex

*number* is a market/group/industry/sector/watchlist number:

0..255 for categoryMarket, categoryGroup, categoryIndustry

0..63 for categorySector

no limit for categoryWatchlist.

ignored for categoryFavorite, categoryIndex

**EXAMPLE**     `// the code adds symbols with last day volume > 100000`  
                  `// to the watch list number 1`  
                  `if( LastValue( V ) > 100000 )`  
                  `{`  
                      `CategoryAddSymbol( "", categoryWatchlist, 1 );`  
                  `}`

**SEE ALSO**     [CategoryGetName\(\)](#) function , [CategoryGetSymbols\(\)](#) function

**References:**

The **CategoryAddSymbol** function is used in the following formulas in AFL on–line library:

- [Calculate composites for tickers in list files](#)
- [WLBuildProcess](#)

**More information:**

[Updated on–line reference](#)

**CATEGORYFIND****Information / Categories****– search for category by name**

(AFL 3.0)

**SYNTAX**      *CategoryFind( "name", category )***RETURNS**      NUMBER**FUNCTION**      It allows to search for category by name. It takes category name and kind as parameters and returns INDEX (ordinal number).

For example it allows to find watch list index by name. The index (in the example below watch list number) can be later used in functions that need the index (like CategoryGetSymbols).

**EXAMPLE**      `wlnumber = CategoryFind( "MyWatch List 1", categoryWatchlist );`  
                 `mysymbols = CategoryGetSymbols( categoryWatchlist, wlnumber );`

**SEE ALSO**      [CategoryAddSymbol\(\)](#) function , [CategoryGetName\(\)](#) function , [CategoryGetSymbols\(\)](#) function , [CategoryRemoveSymbol\(\)](#) function

**References:**

The **CategoryFind** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

## CATEGORYGETNAME

– get the name of a category

Information / Categories  
(AFL 2.5)

**SYNTAX**      *CategoryGetName( category, number)*

**RETURNS**     STRING

**FUNCTION**    The **CategoryGetName** function retrieves the name of category.

*category* is one of the following:

- categoryMarket
- categoryGroup
- categorySector
- categoryIndustry
- categoryWatchlist
- categoryFavorite
- categoryIndex

*number* is a market/group/industry/sector/watchlist number:  
0..255 for categoryMarket, categoryGroup, categoryIndustry  
0..63 for categorySector  
no limit for categoryWatchlist.  
ignored for categoryFavorite, categoryIndex

### EXAMPLE

```
CategoryGetName( categoryWatchlist, 1 );
```

**SEE ALSO**     [CategoryGetSymbols\(\)](#) function

### References:

The **CategoryGetName** function is used in the following formulas in AFL on–line library:

### More information:

[Updated on–line reference](#)

**CATEGORYGETSYMBOLS**

– retrieves comma-separated list of symbols belonging to given category

Information /  
Categories  
(AFL 2.5)

**SYNTAX**      *CategoryGetSymbols( category, index )*

**RETURNS**    STRING

**FUNCTION**    Retrieves comma-separated list of symbols belonging to given category Supported categories:

- categoryMarket
- categoryGroup
- categorySector
- categoryIndustry
- categoryWatchlist
- categoryFavorite
- categoryIndex

index is a market/group/industry/sector/watchlist number:

0..255 for categoryMarket, categoryGroup, categoryIndustry

0..63 for categorySector

no limit for categoryWatchlist.

ignored for categoryFavorite, categoryIndex

Use **StrExtract** function to extract individual symbols from the list.

**EXAMPLE**

```

/* note: if given watch list contains lots of symbols
** performance may be poor
** AVOID SUCH CODES IN REAL-TIME MODE !
*/
function CreateAverageForWatchList( listnum )
{
    // retrieve comma-separated list of symbols in watch list
    list = CategoryGetSymbols( categoryWatchlist, listnum );

    Average = 0; // just in case there are no watch list members

    for( i = 0; ( sym = StrExtract( list, i ) ) != ""; i++ )
    {
        f = Foreign( sym, "C" );
        if( i == 0 ) Average = f;
        else Average = Average + f;
    }
    return Average / i; // divide by number of components
}

Plot( CreateAverageForWatchList( 1 ), "Avg of WL 1", colorGreen );

```

**SEE ALSO**    [GETCATEGORYSYMBOLS\(\)](#) function , [StrExtract\(\)](#) function , [INWATCHLIST\(\)](#) function , [CategoryGetName\(\)](#) function

## References:

The **CategoryGetSymbols** function is used in the following formulas in AFL on–line library:

- [Calculate composites for tickers in list files](#)
- [Relative Strength](#)

## More information:

[Updated on–line reference](#)



**CATEGORYREMOVESYMBOL**

Information / Categories

**– remove a symbol from a category**

(AFL 2.5)

**SYNTAX**     *CategoryRemoveSymbol( symbol, category, number )***RETURNS**    NOTHING**FUNCTION**   Removes the *symbol* from given *category*.

Note that for markets, groups, industries 'removing' means moving from given category to category with number zero, since the symbol is assigned always to one and only one market, group, industry and sector. This limitation does not apply to watchlists, favorites, and index categories. When symbol string is empty ("" ) then current symbol is used.

*category* is one of the following:

- categoryMarket
- categoryGroup
- categorySector
- categoryIndustry
- categoryWatchlist
- categoryFavorite
- categoryIndex

*number* is a market/group/industry/sector/watchlist number:  
 0..255 for categoryMarket, categoryGroup, categoryIndustry  
 0..63 for categorySector  
 no limit for categoryWatchlist.  
 ignored for categoryFavorite, categoryIndex

**EXAMPLE**     `// the code removes the symbols with last day volume < 1000`  
                  `// from the watch list number 1`  
                  `if( LastValue( V ) < 1000 )`  
                  `{`  
                      `CategoryRemoveSymbol( "", categoryWatchlist, 1 );`  
                  `}`

**SEE ALSO**     [CategoryAddSymbol\(\)](#) function , [CategoryGetName\(\)](#) function , [CategoryGetSymbols\(\)](#) function

**References:**

The **CategoryRemoveSymbol** function is used in the following formulas in AFL on–line library:

- [Calculate composites for tickers in list files](#)
- [WLBUILDPROCESS](#)

**More information:**

[Updated on–line reference](#)

**CCI****Indicators****– commodity channel index**

**SYNTAX**      **CCI( periods = 14 )**  
                  **CCla( array, periods = 14 )**

**RETURNS**    ARRAY

**FUNCTION**    Calculates the Commodity Channel Index (using *periods* averaging range ).  
 Second version (CCla) accepts input array, so CCI can be applied to array different than close. (CCla exists in AFL 2.2+ only (v.4.20+))

**EXAMPLE**    CCI( 14 )  
                  CClA( High, 14 );

**SEE ALSO****Comments:**

<b>Tomasz Janeczko</b> tj —at— amibroker.com 2003-09-03 04:24:35	<p>CCI uses internally 'Avg' built-in price array.          'Avg' is also known as typical price:  <math display="block">\text{Avg} = ( H + L + C ) / 3</math></p> <p>So          CCI( period ) is equivalent to CClA( Avg, period ).</p> <p>Therefore if you want to calculate CCI from Foreign ticker you should overwrite Avg array,          instead of OHLC:</p> <p><math display="block">\text{Avg} = ( \text{Foreign}("!VIX", "H") + \text{Foreign}("!VIX", "L") + \text{Foreign}("!VIX", "C") ) / 3;</math> <math display="block">\text{cc} = \text{CCI}(\text{period});</math></p> <p>Alternatively use CClA that takes array directly:</p> <p><math display="block">\text{cc} = \text{CCla}( \text{Foreign}("!VIX", "H") + \text{Foreign}("!VIX", "L") + \text{Foreign}("!VIX", "C") ) / 3, \text{period});</math></p>
---	--

**References:**

The **CCI** function is used in the following formulas in AFL on-line library:

- [Adaptave Zones O/B &O/S Oscillator](#)
- [Auto-Optimization Framework](#)
- [CCI Woodies Style](#)
- [CCI\(20\) Divergence Indicator](#)
- [CCI/DI+- COMBO indicator](#)
- [Commodity Channel Index](#)
- [Price with Woodies Pivots](#)
- [RSI "based" Trading System](#)
- [RSIS](#)
- [Smoothed RSI Buy Signals](#)

- [The Stochastic CCI](#)
- [Woodies CCI](#)

**More information:**

[Updated on-line reference](#)

**CEIL****Math functions****– ceil value**

**SYNTAX**      *ceil( number )*  
                 *ceil( array )*

**RETURNS**    NUMBER,  
                 ARRAY

**FUNCTION**    Calculates the lowest integer that is greater than NUMBER or ARRAY.

**EXAMPLE**     The formula *ceil( 6.2 )* returns 7; the formula *ceil(-6.2)* returns -6.

**SEE ALSO**     [FLOOR\(\)](#) function , [INT\(\)](#) function , [ROUND\(\)](#) function

**References:**

The **ceil** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [Bullish Percent Index 2 files combined](#)
- [Bullish Percent Index 2004](#)
- [N–period candlesticks \(time compression\)](#)
- [PFChart – High/Low prices Sept2003](#)
- [PFchart with range box sizes](#)
- [PF Chart – Close – April 2004](#)
- [Renko Chart](#)
- [Triangle exploration using PFChart](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**

[Updated on–line reference](#)

**CHAIKIN****Indicators****– chaikin oscillator**

**SYNTAX**      *chaikin( fast = 9, slow = 14 )*

**RETURNS**    ARRAY

**FUNCTION**    Calculates the Chaikin Oscillator with averaging parameters: *fast*, *slow*

**EXAMPLE****SEE ALSO****References:**

The **Chaikin** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**CLIPBOARDGET****– retrieves current contents of Windows clipboard****Miscellaneous functions**  
(AFL 2.60)**SYNTAX**     *ClipboardGet()***RETURNS**    STRING**FUNCTION**    Retrieves current contents of Windows clipboard**EXAMPLE**

```
"Contents of the Windows clipboard" + ClipboardGet();
```

**SEE ALSO**    [ClipboardSet\(\)](#) function**References:**

The **ClipboardGet** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**CLIPBOARDSET****Miscellaneous functions****– copies the text to the Windows clipboard**

(AFL 2.6)

**SYNTAX**     *ClipboardSet( "Text" );***RETURNS**    NUMBER**FUNCTION**    Copies the "text" to the Windows clipboard.  
Returns True (1) on success and 0 on failure

**EXAMPLE**     *// this can be used to put dynamically-constructed texts into*  
                 *// clipboard*  
                 *//*  
                 *ClipboardSet( "The price of " + FullName() + " is " + Close );*

**SEE ALSO**     *ClipboardGet()* function**References:**

The **ClipboardSet** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**COLORHSB****Miscellaneous functions****– specify color using Hue–Saturation–Brightness**

(AFL 2.80)

**SYNTAX**     *ColorHSB( hue, saturation, brightness )***RETURNS**    NUMBER**FUNCTION**    The function allows to specify color out of 16 million color (24 bit) palette using Hue, Saturation and Brightness parameters.

The return value is a number that can be used in Plot, PlotOHLC, PlotForeign, AddColumn, AddTextColumn functions to specify chart or column color.

Parameters:

- **hue** – represents gradation of color within the optical spectrum (as in rainbow)
- **saturation** represents "vibrancy" of the color
- **brightness** represents brightness.

Each parameter ranges from 0 to 255, where 0 represents 0% saturation/brightness or 0 degree hue in HSV color wheel, and 255 represents 100% saturation/brightness or 360degrees hue in HSV color wheel.

When you modify hue from 0 to 255 you will see consecutive rainbow colors starting from red, through yellow and green to blue and violet.

For more information about HSB color space please read:

[http://en.wikipedia.org/wiki/HSB\\_color\\_space](http://en.wikipedia.org/wiki/HSB_color_space)

**EXAMPLE**

```
// Example 1:
// 3-d multicolor multiple moving average cloud chart
side = 1;
increment = Param("Increment",2, 1, 10, 1 );
for( i = 10; i < 80; i = i + increment )
{
    up = MA( C, i );
    down = MA( C, i + increment );

    if( ParamToggle("3D effect?", "No|Yes" ) )
        side = IIf(up<=down AND Ref( up<=down, 1 ), 1, 0.6 );

    PlotOHLC( up,up,down,down, "MA"+i, ColorHSB( 3*(i - 10),
        Param("Saturation", 128, 0, 255 ),
        side * Param("Brightness", 255, 0, 255 ) ), styleCloud |
styleNoLabel );
}

// Example 2:
////////
//Color-parade exploration
```



```
Filter=1;  
for( i = 0; i < 256; i = i + 16 )  
    AddColumn( C, "C", 1.2, colorDefault, ColorHSB( ( BarIndex() + i )  
    % 256, 255-i, 255 ) );
```

**SEE ALSO** [ColorRGB\(\)](#) function , [PLOT\(\)](#) function , [AddColumn\(\)](#) function

#### References:

The **ColorHSB** function is used in the following formulas in AFL on-line library:

#### More information:

[Updated on-line reference](#)

**COLORRGB****Miscellaneous functions****– specify color using Red–Green–Blue components**

(AFL 2.80)

**SYNTAX**     *ColorRGB( red, green, blue )***RETURNS**     NUMBER**FUNCTION**     The function allows to specify color out of 16 million color (24 bit) palette using Red, Green, Blue components.

The return value is a number that can be used in Plot, PlotOHLC, PlotForeign, AddColumn, AddTextColumn functions to specify chart or column color.

Parameters:

red, green, blue – represent color component values in range 0..255 each

For more information about RGB color model please read:

[http://en.wikipedia.org/wiki/RGB\\_color\\_model](http://en.wikipedia.org/wiki/RGB_color_model)

**EXAMPLE**     Plot( MA(C,10), "Light Red", ColorRGB( 255, 128, 128 ) ); Plot( MA(C,20), "Light Green", ColorRGB( 128, 255, 128 ) ); Plot( MA(C,30), "Light Blue", ColorRGB( 128, 128, 255 ) );

**SEE ALSO**     [ColorHSB\(\)](#) function , [PLOT\(\)](#) function , [AddColumn\(\)](#) function

**References:**

The **ColorRGB** function is used in the following formulas in AFL on–line library:

- [Hurst "Like" DE](#)

**More information:**

[Updated on–line reference](#)

**CORRELATION****Statistical functions****– correlation**

(AFL 1.4)

**SYNTAX**     *correlation( ARRAY1, ARRAY2, periods )***RETURNS**     ARRAY**FUNCTION**     Calculates correlation between ARRAY1 and ARRAY2 using *periods* range

For more information about correlation please check this:

<http://en.wikipedia.org/wiki/Correlation>

**EXAMPLE**     *// the line below calculates*  
                  *// Correlation between Close price AND AND Close price 5 days back*  
                  *Correlation( Close, Ref( Close, -5 ), 5 );*

*// Built-in correlation can be re-coded with*  
*// basic AFL functions like MA (moving average) - which*  
*// is equivalent for "expected value" statistic term*  
*// and few basic arithmetic operations*

```
function Correl( x, y, number )
{
  nom= MA( x * y, number ) - MA( x, number ) * MA( y, number );
  denom = sqrt( MA( x ^ 2, number ) - MA( x, number ) ^ 2 ) *
  sqrt( MA( y ^ 2, number ) - MA( y, number ) ^ 2 );
  return nom/denom;
}
```

```
Graph0=Correlation( C, Ref( H, -2 ), 10 ); // built-in
```

```
Graph1=Correl( C, Ref( H, -2 ), 10 ); // re-coded;
```

**SEE ALSO****References:**The **Correlation** function is used in the following formulas in AFL on-line library:

- Alpha and Beta and R\_Squared Indicator
- crMathLib
- Cycle Highlighter
- Cycle Highlighter (auto best-fit)
- R-Squared

**More information:**[Updated on-line reference](#)

**COS****Math functions****– cosine**

**SYNTAX**      *cos( NUMBER )*  
                 *cos( ARRAY )*

**RETURNS**    NUMBER  
                 ARRAY

**FUNCTION**   Returns the cosine of NUMBER or ARRAY. Assumes that the NUMBER or ARRAY values are in radians.

**EXAMPLE**    *cos( C )*

**SEE ALSO**    [atan\(\)](#) function , [SIN\(\)](#) function

**References:**

The **cos** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [Luna Phase](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**

[Updated on–line reference](#)

## **COSH**

### **– hyperbolic cosine function**

**Math functions**  
(AFL 2.80)

**SYNTAX**      *cosh( NUMBER )*  
                 *cosh( ARRAY )*

**RETURNS**    NUMBER,  
                 ARRAY

**FUNCTION**   Returns the hyperbolic cosine of NUMBER or ARRAY. This function assumes that the ARRAY values are in radians.

#### **EXAMPLE**

#### **SEE ALSO**

#### **References:**

The **cosh** function is used in the following formulas in AFL on–line library:

#### **More information:**

[Updated on–line reference](#)

## CREATEOBJECT

### – create COM object

Miscellaneous functions  
(AFL 1.8)

**SYNTAX**     *createobject*(

**RETURNS**    OBJECT

**FUNCTION**    Creates the instance of "Server.Class" COM object. The return value should be assigned to a variable that is used latter for calling the methods of the object.

Note: this function creates the instance of the object everytime the formula is executed (the object is released automatically at the end of the formula – no explicit freeing is necessary)

**EXAMPLE**     myobj = CreateObject("MyOwnActiveX.Class1");  
                 myobj.Method( 1, 2, Close ); // call the method of myobj COM object

**SEE ALSO**     [CREATESTATICOBJECT\(\)](#) function

#### References:

The **CreateObject** function is used in the following formulas in AFL on–line library:

- [AFL–Excel](#)
- [Auto Export to Gif](#)
- [Calculate composites for tickers in list files](#)
- [SectorRSI](#)

#### More information:

[Updated on–line reference](#)

**CREATESTATICOBJECT**  
– create static COM object**Miscellaneous functions**  
(AFL 1.8)**SYNTAX**     *createstaticobject*(**RETURNS**    OBJECT

**FUNCTION**    Creates the single static instance (one per AmiBroker session) of "Server.Class" COM object. The return value should be assigned to a variable that is used latter for calling the methods of the object. This function is useful for "heavyweight" COM object like QuotesPlus ActiveX for example.

Note: this function creates the instance of the object only once when the formula is executed for the first time. Then the object is cached internally for all consecutive calls. It is also shared if multiple formulas use the same object using CreateStaticObject call. The object is automatically released when AmiBroker is closed.

**EXAMPLE**     myobj = CreateStaticObject("MyOwnActiveX.Class1");  
myobj.Method( 1, 2, Close ); // call the method of myobj COM object

**SEE ALSO**     [CreateObject\(\)](#) function

**References:**

The **CreateStaticObject** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**CROSS****Trading system toolbox****– crossover check****SYNTAX**      **Cross( ARRAY1, ARRAY2 )****RETURNS**      ARRAY

**FUNCTION**      Gives a "1" or true on the day that ARRAY1 crosses above ARRAY2. Otherwise the result is "0".  
                     To find out when ARRAY1 crosses **below** ARRAY2, use the formula cross(ARRAY2, ARRAY1)

**EXAMPLE**      cross( close, ema(close,9) )**SEE ALSO****References:**

The **Cross** function is used in the following formulas in AFL on–line library:

- 'D/9E H'
- 'D/9E H'
- 'DE'H37href='http://www.amibroker.com/library/detail.php?id=718target='\_blank'>abosliman2005m
- AFL Example
- AFL Example – Enhanced
- AFL–Excel
- Against all odds
- AR\_Prediction.afl
- ATR Study
- Auto–Optimization Framework
- Bow tie
- Bull Fear / Bear Fear
- CandleStick Comentary---Help needed
- CCI(20) Divergence Indicator
- Chandelier Exit
- Chandelier Exit or Advanced Trailing Stop
- Commodity Channel Index
- Customised Avg. Profit %, Avg. Loss % etc
- DateNum\_DateStr
- Demand Index
- Dinapoli Guru Commentary
- DMI Spread Index
- Dynamic Momentum Index
- Dynamic Momentum Index
- Ed Seykota's TSP: EMA Crossover System
- ekeko price chart
- Ema bands
- EMA Crossover
- FastStochK FullStochK–D
- Follow the Leader
- Fund Screener
- Gann HiLo Indicator and System
- hassan



- Indicator Explorer (ZigZag)
- Lagging MA–Xover
- MACD and histogram divergence detection
- MACD commentary
- MACD optimize
- Performance Check
- prakash
- Relative Strength Index
- RSI "based" Trading System
- RUTVOL timing signal with BB Scoring routine
- Sainath Sidgiddi
- Smoothed RSI Buy Signals
- STO &MACD Buy Signals with Money–Management
- Stochastic Fast%K and Full
- Stochastic optimize
- Stochastics Trendlines
- Stock price AlertIf
- Stops Implementation in AFS
- Support and Resistance
- The D\_oscillator
- Trend Analysis\_Comentary
- Trend exploration with multiple timeframes
- Trend Trigger Factor
- Triangular Moving Average new
- Trigonometric Fit – TrigFit with AR for cos / sin
- Using From and To dates from Auto Analysis in Code
- VAMA
- Visualization of stoploses and profit in chart
- Volatility Breakout with Bollinger Bands
- Volume Oscillator
- Williams %R with 9 period signal line
- Williams Alligator system

**More information:**

Updated on–line reference

**CUM****Moving averages, summation****– cumulative sum****SYNTAX**     *cum( ARRAY )***RETURNS**     ARRAY**FUNCTION**     Calculates a cumulative sum of the ARRAY from the first period in the chart.**EXAMPLE**     The formula `cum( 1 )` calculates an indicator that rises one point for each day since the beginning of the chart – this is an equivalent of bar number – especially useful if you want to detect the last bar: `ThisIsLastBar = cum( 1 ) == lastvalue( cum( 1 ) );`**SEE ALSO**     [SUM\(\)](#) function**Comments:**

<p><b>Graham Kavanagh</b>  gkavanagh@e-wire.net.au  2004-08-09 07:49:35</p>	<p>Sum adds up the last "n" number of bars. It sums whatever you put into the first part of the sum formula.</p> <p>Cum(1) adds 1 to the previous value of Cum, so the first bar is 1 and it just keeps adding one to the last bar value of cum(1).  You can use Cum to add anything, like how many times you get rising days in the entire chart:</p> <p>Rise = C&gt;O; //this gives results of 0 or 1  TotalRise = Cum(Rise);</p> <p>You could limit this as well to time periods, or any other condition  Example would be one for total rise days since 1995:</p> <p>RecentRise = C&gt;O and Year()&gt;=1995; //this gives results of 0 or 1  TotalRise = Cum(RecentRise);</p> <p>If you wanted to know how many rising days in the last 12 bars you would use:</p> <p>LastRises = Sum(Rise,12);</p> <p>Hope this helps</p>
---	---

**References:**

The **Cum** function is used in the following formulas in AFL on-line library:

- ['D/9E H'](#)
- ['D/9E H'](#)
- [AR\\_Prediction.afl](#)
- [Auto-Optimization Framework](#)
- [Automatic Trend-line](#)
- [Baseline Relative Performance Watchlist charts V2](#)
- [Buff Volume Weighted Moving Averages](#)

- Bullish Percent Index 2 files combined
- Candle Stick Analysis
- Color Display.afl
- Cycle Highlighter
- Cycle Highlighter (auto best-fit)
- Dave Landry PullBack Scan
- Demand Index
- Double top detection
- ekeko price chart
- Fund Screener
- Gann Swing Chart
- Head & Shoulders Pattern
- Hurst Constant
- Linear Regression Line & Bands
- Linear Regression Line w/ Std Deviation Channels
- Market Profile & Market Volume Profile
- McClellan Summation Index
- Modified Momentum Finder DDT-NB
- Monthly bar chart
- Moving Average "Crash" Test
- Multiple sinus noised
- N-period candlesticks (time compression)
- nth ( 1 – 8 ) Order Polynomial Fit
- Pattern Recognition Exploration
- Performance Overview
- Price Persistency
- Projection Oscillator
- QP2 Float Analysis
- R-Squared
- Random Walk
- Regression Analysis Line
- RSI Trendlines and Wedges
- RSIS
- Steve Woods' Cum. Vol. Float + Cum. Vol. Channels
- Steve Woods' Cumulative Vol. Percentage Indicator
- Stochastics Trendlines
- Triangle exploration using PFChart
- Triangular Moving Average
- Triangular Moving Average new
- Trigonometric Fit – TrigFit with AR for cos / sin
- Volatility Quality Index
- Weekly chart
- Williams Alligator system

#### More information:

Updated on-line reference

**DATE****Date/Time****– date**

(AFL 1.1)

**SYNTAX**     **date()****RETURNS**     STRING**FUNCTION**     It is used to display the selected date in commentary / interpretation window**EXAMPLE**     date()**SEE ALSO****References:**

The **Date** function is used in the following formulas in AFL on–line library:

- 'D/9E H'
- 'D/9E H'
- AB system full automation scripts
- AccuTrack
- ADX Indicator – Colored
- AFL Example
- AFL Example – Enhanced
- Against all odds
- Another Fib Level
- ATR Trading System
- Bullish Percent Index 2004
- Candle Stick Demo
- CCI Woodies Style
- Color Coded Short Term Reversal Signals
- Commodity Channel Index
- DateNum\_DateStr
- Dinapoli Guru Commentary
- Double Smoothed Stochastic from W.Bressert
- Elder safe Zone Long + short
- Elder Triple Screen Trading System.
- Elder's SafeZone Stop
- ElderSafeZoneStopLong
- ElderSafeZoneStopShort
- EMA Crossover Price
- Futures – Dollar Move Indicator
- Futures – Dollar Move Today Indicator
- hassan
- lastNDaysBeforeDate
- MA Difference 20 Period
- MACD commentary
- MACD Histogram – Change in Direction
- Main price chart with Rainbow & SAR
- PF Chart – Close – April 2004
- Pivot Point and Support and Resistance Points
- Pivots for Intraday Forex Charts
- Position Sizing and Risk Price Graph

- [Position Sizing and Risk Price Graph – 2](#)
- [Price with Woodies Pivots](#)
- [Probability Calculator](#)
- [Relative Strength Index](#)
- [Relative Strength Multichart of up to 10 tickers](#)
- [Shares To Buy Price Graph](#)
- [Support Resistance levels](#)
- [Trading ATR 10–1](#)
- [Triangular Moving Average new](#)
- [William's % R](#)
- [Williams %R with 9 period signal line](#)
- [Woodies CCI](#)
- [Zig Zag Indicator with Valid Entry and Exit Points](#)

**More information:**

[Updated on–line reference](#)

**DATENUM****Date/Time****– date number**

(AFL 1.4)

**SYNTAX**     *datenum()***RETURNS**     ARRAY

**FUNCTION**     Returns the array with numbers that represent quotation dates coded as follows:  
                     10000 \* (year – 1900) + 100 \* month + day, so 2001–12–31 becomes 1011231 and  
                     1995–12–31 becomes 951231

**EXAMPLE**     *datenum();***SEE ALSO****References:**

The **DateNum** function is used in the following formulas in AFL on–line library:

- [Bullish Percent Index 2004](#)
- [DateNum\\_DateStr](#)
- [Date\\_To\\_Num\(\), Time\\_To\\_Num\(\)](#)
- [Fund Screener](#)
- [Gordon Rose](#)
- [lastNDaysBeforeDate](#)
- [PF Chart – Close – April 2004](#)
- [Rea Time Daily Price Levels](#)
- [Steve Woods' Cum. Vol. Float + Cum. Vol. Channels](#)
- [Steve Woods' Cumulative Vol. Percentage Indicator](#)
- [TWS auto–export Executions–file parser](#)
- [Weekly chart](#)
- [Zig Zag Indicator with Valid Entry and Exit Points](#)

**More information:**

[Updated on–line reference](#)

**DATETIME****Date/Time****– retrieves encoded date time**

(AFL 2.3)

**SYNTAX**     *DateTime()***RETURNS**     ARRAY**FUNCTION**     Returns array of encoded date/time values suitable for using with AddColumn and formatDateTime constant to produce date time formatted according to your system settings.**EXAMPLE**     1. Simple date/time column

```
AddColumn( DateTime(), "Date / Time", formatDateTime );
```

2. Example (produces signal file accepted by various other programs):

```
Buy=Cross(MACD(),Signal());
Sell=Cross(Signal(), MACD());
Filter=Buy OR Sell;
SetOption("NoDefaultColumns", True );
AddColumn( DateTime(), "Date", formatDateTime );
AddColumn( IIf( Buy, 66, 83 ), "Signal", formatChar );
```

**SEE ALSO****References:**

The **DateTime** function is used in the following formulas in AFL on–line library:

- [Ed Seykota's TSP: EMA Crossover System](#)
- [Ed Seykota's TSP: Support and Resistance](#)
- [Futures – Dollar Move Indicator](#)
- [Gordon Rose](#)
- [PF Chart – Close – April 2004](#)
- [Pivot Finder](#)
- [Relative Strength Multichart of up to 10 tickers](#)

**More information:**

[Updated on–line reference](#)

## DATETIMECONVERT

### – date/time format conversion

**Date/Time**  
(AFL 2.90)

**SYNTAX**     *DateTimeConvert( format, date, time = Null )*

**RETURNS**    NUMBER or ARRAY

**FUNCTION**    The function allows to convert from DateTime format to DateNum and TimeNum and vice versa.

format parameter controls the direction of conversion:

- format = 0 – converts DateTime format to DateNum format, example

mydatenum = DateTimeConvert( 0, DateTime() ); // – this returns DateNum  
date argument should be in datetime format, time argument in this case should not be used

- format = 1 – converts DateTime format to TimeNum format, example:

mytimenum = DateTimeConvert( 1, DateTime() ); // – returns timenum  
date argument should be in datetime format, time argument in this case should not be used

- format = 2 – converts from DateNum and optionally TimeNum to DateTime format, example:

mydatetime = DateTimeConvert( 2, DateNum(), TimeNum() );  
date argument should be in datenum format, time argument (optional) should be in timenum format. In case of EOD data you can skip time argument:

mydatetime = DateTimeConvert( 2, DateNum() );

- format = 3 – (AB5.0 or higher) converts DateTime format to Seconds (00..59) example:

myseconds = DateTimeConvert( 3, DateTime() );

date argument should be in datetime, format, time argument in this case should not be used

- format = 4 – (AB5.0 or higher) converts DateTime format to Minutes(00..59) example:

myminute = DateTimeConvert( 4, DateTime() );

date argument should be in datetime, format, time argument in this case should not be used

- format = 5 – (AB5.0 or higher) converts DateTime format to Hours (00..23) example:

myhour = DateTimeConvert( 5, DateTime() );

date argument should be in datetime, format, time argument in this case should not be used



**EXAMPLE**

```
mydatennum = DateTimeConvert( 0, DateTime() ); // - this returns
DateNum
mytimenum = DateTimeConvert( 1, DateTime() ); // - returns timenum
mydatetime = DateTimeConvert( 2, DateNum(), TimeNum() );
mydatetime = DateTimeConvert( 2, DateNum() );
```

**SEE ALSO** [DateNum\(\)](#) function , [DateTime\(\)](#) function , [DateTimeToStr\(\)](#) function , [Day\(\)](#) function , [DayOfWeek\(\)](#) function , [DayOfYear\(\)](#) function , [TIMENUM\(\)](#) function , [MONTH\(\)](#) function , [YEAR\(\)](#) function , [HOUR\(\)](#) function , [MINUTE\(\)](#) function , [SECOND\(\)](#) function

**References:**

The **DateTimeConvert** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)

## **DATETIMETOSTR** – convert datetime to string

**String manipulation**  
(AFL 2.8)

**SYNTAX**     *DateTimeToStr( NUMBER )*

**RETURNS**    STRING

**FUNCTION**    Converts number representing date/time value (for example obtained using GetCursorXPosition(), Now(), DateTime() functions) to the corresponding STRING (text).

**EXAMPLE**     `ToolTip="X="+DateTimeToStr( GetCursorXPosition( ) )  
                  + "nY="+GetCursorYPosition( ) ;`

**SEE ALSO**     [DATETIME\(\)](#) function , [NOW\(\)](#) function , [StrToDateTime\(\)](#) function

### **References:**

The **DateTimeToStr** function is used in the following formulas in AFL on–line library:

### **More information:**

[Updated on–line reference](#)

**DAY**  
**– day of month****Date/Time**  
(AFL 1.4)**SYNTAX**     *day()***RETURNS**     ARRAY**FUNCTION**     Returns the array with days (1–31)**EXAMPLE**     writeif( day() < 3, "Beginning of the month", "The rest of the month" );**SEE ALSO****References:**

The **Day** function is used in the following formulas in AFL on–line library:

- [Days to Third Friday](#)
- [Expiry Thursday for Indian markets](#)
- [Export Intraday Data](#)
- [IntraDay Open Marker](#)
- [Luna Phase](#)
- [LunarPhase](#)
- [Market Profile &Market Volume Profile](#)
- [N–period candlesticks \(time compression\)](#)
- [Option Calls, Puts and days till third friday.](#)
- [Plot Monthly,Weekly and Daily Moving average](#)
- [Relative Strength Multichart of up to 10 tickers](#)

**More information:**

[Updated on–line reference](#)

## DAYOFWEEK

– day of week

**Date/Time**  
(AFL 1.4)

**SYNTAX**     *dayofweek()*

**RETURNS**    ARRAY

**FUNCTION**   Returns the array with day of week (0–6):  
                  0 – Sunday  
                  1 – Monday  
                  ...  
                  5 – Friday  
                  6 – Saturday

**EXAMPLE**    buy = dayofweek() == 1; // buy on Monday  
                  sell = dayofweek() == 5; // sell on Friday

### SEE ALSO

#### References:

The **DayOfWeek** function is used in the following formulas in AFL on–line library:

- [Days to Third Friday](#)
- [Ed Seykota's TSP: EMA Crossover System](#)
- [Expiry Thursday for Indian markets](#)
- [Option Calls, Puts and days till third friday.](#)
- [Plot Monthly, Weekly and Daily Moving average](#)
- [RSI of Weekly Price Array](#)
- [Sainath Sidgiddi](#)
- [Stochastic of Weekly Price Array](#)
- [Time Frame Weekly Bars](#)
- [Weekly chart](#)
- [Weekly Trend in Daily Graph](#)

#### More information:

[Updated on–line reference](#)

## DAYOFYEAR

– get ordinal number of day in a year

**Date/Time**  
(AFL 2.4)

**SYNTAX**     *DayOfYear()*

**RETURNS**     ARRAY

**FUNCTION**     Returns the calendar day number counting from beginning of the year January 1st is 1.  
Maximum number returned is 366

**EXAMPLE**     `Filter=1;  
AddColumn(DayOfYear(),"Day of Year");`

**SEE ALSO**     [DAYOFWEEK\(\)](#) function

### References:

The **DayOfYear** function is used in the following formulas in AFL on–line library:

- [End Of Year Trading](#)

### More information:

[Updated on–line reference](#)

**DECISSUES****Composites****– declining issues**

(AFL 1.2)

**SYNTAX**     *DeclIssues()***RETURNS**     ARRAY**FUNCTION**     Returns the number of declining issues for a given market (the one that currently analysed stock belongs to)**EXAMPLE**     DeclIssues()**SEE ALSO****References:**

The **DeclIssues** function is used in the following formulas in AFL on–line library:

- [Absolute Breadth Index](#)
- [Breadth Thrust](#)
- [McClellan Oscillator](#)
- [McClellan Summation Index](#)

**More information:**

[Updated on–line reference](#)

**DECVOLUME**  
**– declining issues volume****Composites**  
(AFL 1.2)**SYNTAX**     *DecVolume()***RETURNS**     ARRAY**FUNCTION**     Returns the volume of declining issues for a given market (the one that currently analysed stock belongs to)**EXAMPLE**     DecVolume()**SEE ALSO****References:**

The **DecVolume** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**DEMA****– double exponential moving average****Moving averages, summation**  
(AFL 2.0)**SYNTAX**     *dema( ARRAY, periods )***RETURNS**     ARRAY**FUNCTION**     Calculates double exponentially smoothed average – DEMA. The function accepts time-variable *periods*.**EXAMPLE**     DEMA( Close, 5 )**SEE ALSO**     MA(), EMA(), WMA(), TEMA()**Comments:**

<b>Tomasz Janeczko</b> tj ---at--- amibroker.com 2003-02-06 13:51:31	DEMA can be implemented via EMA:  Len=10; Graph0= 2 * EMA( C, len ) – EMA( EMA( C, len ), Len );  // for comparison only Graph1=DEMA(C,Len);
<b>Tomasz Janeczko</b> tj ---at--- amibroker.com 2003-04-27 15:43:17	Note to the comment above: EMA and DEMA use different initialization method. DEMA[ 0 ] is initialized with first value of input array, while EMA[ len ] is initialized with simple moving average to match output with Metastock. Therefore they will converge at 2 * Len bars from Graph0 start ( 6 * Len bars since beginning of the data).
<b>Tomasz Janeczko</b> tj ---at--- amibroker.com 2003-04-27 15:48:11	DEMA can also be implemented using new for looping:  Len = 20; Plot( DEMA( Close, Len ), "Built-in DEMA", colorRed );  factor = 2 / (Len + 1 );  e1 = e2 = Close[ 0 ]; // initialize  for( i = 0; i < BarCount; i++ ) { e1 = factor * Close[ i ] + ( 1 – factor ) * e1; e2 = factor * e1 + ( 1 – factor ) * e2;  myDema[ i ] = 2 * e1 – e2; }  Plot( myDema, "Dema in loop", colorBlue );
<b>Tomasz Janeczko</b> tj ---at--- amibroker.com 2003-04-27 15:51:03	... and can be implemented using AMA:  Len = 20;



	<pre>Factor = 2/(Len+1);  e1 = AMA( Close, Factor ); e2 = AMA( e1, Factor ); Plot( DEMA( Close, Len ), "Built-in DEMA", colorRed ); Plot( 2*e1 - e2, "AMA-implemented DEMA", colorBlue );</pre>
<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2003-04-27 16:26:06	For more information on DEMA see: Stocks & Commodities V. 12:1 (11-19): Smoothing Data With Faster Moving Averages by Patrick G. Mulloy.

**References:**

The **DEMA** function is used in the following formulas in AFL on-line library:

- [Auto-Optimization Framework](#)
- [Dynamic Momentum Index](#)
- [Dynamic Momentum Index](#)
- [Lagging MA-Xover](#)
- [Moving Averages NoX](#)
- [Support Resistance levels](#)
- [The D\\_oscillator](#)
- [The Saturation Indicator D\\_sat](#)
- [Trend Detection](#)

**More information:**

[Updated on-line reference](#)

**EMA****Moving averages, summation****– exponential moving average****SYNTAX**     *ema( ARRAY, periods )***RETURNS**     ARRAY**FUNCTION**     Calculates a *periods* exponential moving average of ARRAY**EXAMPLE**     ema( close, 5 )**SEE ALSO**     [MA\(\)](#) function , [TEMA\(\)](#) function , [AMA\(\)](#) function , [AMA2\(\)](#) function , [DEMA\(\)](#) function , [WMA\(\)](#) function , [WILDERS\(\)](#) function**Comments:**

<b>Nigel Rowe</b> rho@bigpond.com 2003-04-27 18:05:14	See the comments attached to DEMA for a discussion on the differences in the way EMA and others are initialised.  EMA is initialised from a simple MA of equivalent length. (For compatability with some other strange TA software.) The others are initialised from the first value.
--	---

**References:**

The **EMA** function is used in the following formulas in AFL on–line library:

- [accum/dist mov avg crossover SAR](#)
- [AccuTrack](#)
- [ADXbuy](#)
- [AFL Example](#)
- [AFL Example – Enhanced](#)
- [Against all odds](#)
- [Andrews PitchforkV3.3](#)
- [AR\\_Prediction.afl](#)
- [ATR Study](#)
- [Auto–Optimization Framework](#)
- [Balance of Power](#)
- [balance of power](#)
- [BB squeeze](#)
- [BMATRIX Intermediate Term Market Trend Indicator](#)
- [Bollinger – Keltner Bands](#)
- [Bollinger band normalization](#)
- [Breadth Thrust](#)
- [Bull/Bear Volume](#)
- [CCI Woodies Style](#)
- [CCT FibAccordion](#)
- [CCT Kaleidoscope](#)
- [Chaikin's Volatility](#)
- [Compare Sectors against Tickers](#)
- [Coppock Curve](#)
- [Dahl Oscillator modified](#)
- [Dave Landry PullBack Scan](#)
- [Dave Landry Pullbacks](#)

- Demand Index
- Derivative Oscillator
- Divergences
- Double Smoothed Stochastic from W.Bressert
- DT Oscillator
- Dynamic Momentum Index
- Dynamic Momentum Index
- Elder Bear Power
- Elder Bull Power
- Elder Impulse Indicator
- Elder Impulse Indicator V2
- Elder Ray – Bull Bear
- Elder Triple Screen Trading System.
- Elder's Market Thermometer
- Ema bands
- EMA Crossover
- EMA Crossover Price
- Ergodic Oscillator
- Fibonacci Moving averages
- Force index
- Fund Screener
- Hilbert Study
- INTRADAY HEIKIN ASHI new
- Lagging MA–Xover
- MACD and histogram divergence detection
- MACD Histogram – Change in Direction
- Market Direction
- McClellan Oscillator
- McClellan Summation Index
- Moving Averages NoX
- MultiCycle 1.0
- Noor\_Doodie
- Peterson
- Pivots for Intraday Forex Charts
- Plot Monthly, Weekly and Daily Moving average
- Polarized Fractal Efficiency
- Price with Woodies Pivots
- Relative Strength
- Reverse EMA function
- RSI of volume weighted moving average
- RSIS
- RUTVOL timing signal with BB Scoring routine
- Schiff Lines
- Sector Tracking
- SectorRSI
- SF Entry, Stop, PT Indicator
- SIROC Momentum
- Smoothed RSI Buy Signals
- STD\_STK Multi
- STO & MACD Buy Signals with Money–Management
- Stochastic of Weekly Price Array
- StochD\_StochK Single.afl

- [SunI>Support Resistance levels](#)
- [T3](#)
- [T3 Function](#)
- [TAZ Trading Method Exploration](#)
- [The D\\_oscillator](#)
- [The Relative Slope](#)
- [The Relative Slope Pivots](#)
- [The Saturation Indicator D\\_sat](#)
- [tomy\\_frenchy](#)
- [Trend Detection](#)
- [Triangular Moving Average new](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [TRIX](#)
- [Varexlist](#)
- [Vertical Horizontal Filter \(VHF\)](#)
- [Volume Oscillator](#)
- [Weekly Trend in Daily Graph](#)
- [Weinberg's The Range Indicator](#)
- [Williams Alligator system](#)
- [ZeroLag MACD\(p,q,r\)](#)

**More information:**

[Updated on-line reference](#)

**ENABLEROTATIONALTRADING**

Trading system toolbox

**– Turns on rotational–trading mode of the backtester**

(AFL 2.5)

**SYNTAX**     *EnableRotationalTrading()***RETURNS**    NOTHING**FUNCTION**    When placed on the top of system formula it turns on rotational–trading (aka. fund–switching) mode of the backtester.

Note: this function is now marked as obsolete. Use `SetBacktestMode( backtestRotational )` in new formulas.

IMPORTANT NOTE: Unless you specifically want to implement fund–switching/rotational trading system you should NOT use this mode.

Rotational trading is popular method for trading mutual funds. It is also known as fund–switching or scoringIts basic permise is to **rotate symbols all the time** so only top N issues ranked according to some user–definable score are traded. The number of positions open depend on "Max. open positions" setting and available funds / position size. Once position is entered in remains in place until security's rank drops below `WorstRankHeld` (settable via `SetOption("WorstRankHeld", 5 )`). **Regular buy/sell/short/cover signals are not used at all.**

The rotational mode uses only score variable (`PositionScore`) to rank and rotate securities. This idea has been implemented earlier in `PortfolioTrader` AFL formula written by Fred Tonetti with GUI written by Dale Wingo.

To enter this mode you have to call ***EnableRotationalTrading()*** function at the very beginning of your formula. From then on using of buy/sell/short/cover variables is not allowed. Only `PositionScore` variable will be used to rank securities and trade top N securities..

A simple rotational trading formula (stocks with high RSI are best candidates for shorting while stocks with low RSI are best candidates for long positions):

```
EnableRotationalTrading();
SetOption("WorstRankHeld",5);

PositionSize = -25; // invest 25% of equity in single security
PositionScore = 50 - RSI(); // PositionScore has the same meaning as
rScore in PT
```

The score (`PositionScore`) for all securities is calculated first. Then all scores are **sorted according to absolute value of PositionScore**. Then top N are chosen to be traded. N depends on available funds and "max. open positions" setting. Backtester successively enters the trades starting from highest ranked security until the number of positions open reaches "**max. open positions**" or there are no more funds available. The score has the following meaning:

- higher positive score means better candidate for entering long trade

- lower negative score means better candidate for entering short trade
- the score of zero means no trade (exit the trade if there is already open position on given symbol)
- the score equal to **scoreNoRotate** constant means that already open trades should be kept and no new trades entered
- the score equal to **scoreExitAll** constant causes rotational mode backtester to exit all positions regardless of HoldMinBars. Note that this is global flag and it is enough to set it for just any single symbol to exit all currently open positions, no matter on which symbol you use scoreExitAll (it may be even on symbol that is not currently held). By setting PositionScore to scoreExitAll you exit all positions immediately regardless of HoldMinBars setting

***Exits are generated automatically when security's rank drops below "worst rank held".***

There is no real control over when exits happen except of setting low score to force exits. You can also set the score on any (at least one) security to value of scoreNoRotate to prevent rotation (so already open positions are kept). But this is global and does not give you individual control.

**Important:**

**The rotational trading mode uses "buy price" and "buy delay" from the Settings | Trade page as trade price and delay for both entries and exits (long and short)**

**EXAMPLE**    `EnableRotationalTrading();`  
                  `SetOption("WorstRankHeld",5);`

`PositionSize = -25; // invest 25% of equity in single security`  
`PositionScore = 50 - RSI(); // PositionScore has the same meaning as`  
`rScore in PT`

**SEE ALSO**    `SetBacktestMode()` function

**References:**

The **EnableRotationalTrading** function is used in the following formulas in AFL on-line library:

- [Relative Strength](#)

**More information:**

[Updated on-line reference](#)

**ENABLESCRIPT****Miscellaneous functions****– enable scripting engine****SYNTAX**     *EnableScript( "enginename" )***RETURNS**    NOTHING**FUNCTION**    Enables AFL scripting host. *enginename* specifies which scripting language will be used. Allowable values: "jscript", "vbscript".**EXAMPLE**     EnableScript( "jscript" );  
                  EnableScript("vbscript"); **SEE ALSO** [AFL scripting host](#)**SEE ALSO****References:**

The **EnableScript** function is used in the following formulas in AFL on–line library:

- [accum/dist mov avg crossover SAR](#)
- [AFL–Excel](#)
- [AR\\_Prediction.afl](#)
- [Bullish Percent Index 2 files combined](#)
- [danningham penetration](#)
- [Hilbert Study](#)
- [Kagi Chart](#)
- [Monthly bar chart](#)
- [nth \( 1 – 8 \) Order Polynomial Fit](#)
- [PFchart with range box sizes](#)
- [pattern correlation](#)
- [Polarized Fractal Efficiency](#)
- [QP2 Float Analysis](#)
- [Standard Error Bands](#)
- [Steve Woods' Cum. Vol. Float + Cum. Vol. Channels](#)
- [Steve Woods' Float Channel Lines](#)
- [tomy\\_frenchy](#)
- [Trend Continuation Factor](#)
- [Triangle exploration using PFChart](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**

[Updated on–line reference](#)

**ENABLETEXTOUTPUT****– enables/disables text output in the Chart Commentary window****Miscellaneous  
functions  
(AFL 2.2)****SYNTAX**     *EnableTextOutput( enable )***RETURNS**    NOTHING**FUNCTION**    Allows to enable or disable text output in the guru chart commentary window

**EXAMPLE**     `EnableTextOutput(False);`  
                 `variable = "text"; // this won't be written to commentary window`  
                 `EnableTextOutput(True);`

**SEE ALSO**    [\\_N\(\)](#) function**References:**

The **EnableTextOutput** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)



**ENCODECOLOR****Exploration / Indicators****– encodes color for indicator title**

(AFL 2.2)

**SYNTAX**     *EncodeColor( colorIndex )***RETURNS**    STRING

**FUNCTION**    Converts color index to string escape sequence that changes color of text output in chart title. Color escape sequence uses cXX sequence where XX is 2 digit number specifying color index c38 – defines violet, there is a special sequence c–1 that resets to default axis color.

**EXAMPLE**     Title = "This is written in " + **EncodeColor**( colorViolet ) + "violet color " + **EncodeColor**( colorGreen ) + "and this in green"; **SEE ALSO** [Using colors in Indicator Builder](#)

**SEE ALSO****References:**

The **EncodeColor** function is used in the following formulas in AFL on–line library:

- 'R' Channel
- [ADX Indicator – Colored](#)
- [Andrews PitchforkV3.3](#)
- [Another Flb Level](#)
- [ATR Trading System](#)
- [CCI Woodies Style](#)
- [Color Display.afl](#)
- [Cycle Highlighter](#)
- [Cycle Highlighter \(auto best–fit\)](#)
- [EMA Crossover Price](#)
- [FastStochK FullStochK–D](#)
- [Futures – Dollar Move Indicator](#)
- [Futures – Dollar Move Today Indicator](#)
- [Hurst "Like" DE](#)
- [MACD and histogram divergence detection](#)
- [PFchart with range box sizes](#)
- [Position Sizing and Risk Price Graph](#)
- [Position Sizing and Risk Price Graph – 2](#)
- [Price with Woodies Pivots](#)
- [Relative Strength Multichart of up to 10 tickers](#)
- [RSI of Weekly Price Array](#)
- [RUTVOL timing signal with BB Scoring routine](#)
- [Schiff Lines](#)
- [Shares To Buy Price Graph](#)
- [Stochastic of Weekly Price Array](#)
- [Support Resistance levels](#)
- [Trading ATR 10–1](#)
- [Triangular Moving Average](#)
- [Triangular Moving Average new](#)
- [Vertical Horizontal Filter \(VHF\)](#)

**More information:**

[Updated on-line reference](#)

**ENDVALUE****Date/Time**  
(AFL 2.3)**– value of the array at the end of the selected range****SYNTAX**     *EndValue( ARRAY )***RETURNS**    NUMBER**FUNCTION**    This function gives the single value (number) of the ARRAY at the end of the selected range. If no range is marked then the value at the last bar is returned.

To select the range you have to double click in the chart at the beginning of the range and then double click in the chart at the end of the range. Then > < markers will appear above date axis.

**EXAMPLE**    1. Simple commentary:

```
WriteVal( BeginValue( DateTime() ), formatDateTime );
WriteVal( EndValue( DateTime() ), formatDateTime );
"Percentage change of close is " +
WriteVal( 100 * (EndValue( Close ) - BeginValue( Close
))/BeginValue( Close ) ) + "%";
```

2. Get the number of bars in the range and calculate some stats for that range:

```
Period = EndValue( BarIndex() ) - BeginValue( BarIndex() );
StandardDeviationInTheRange = EndValue( StDev( Close, Period ) );
```

**SEE ALSO**    [BEGINVALUE\(\)](#) function**References:**

The **EndValue** function is used in the following formulas in AFL on–line library:

- [Adaptive Price Channel](#)
- [Another Fib Level](#)
- [CAMSLIM Cup and Handle Pattern AFL](#)
- [Futures – Dollar Move Indicator](#)
- [nth \( 1 – 8 \) Order Polynomial Fit](#)
- [pattern correlation](#)
- [Relative Strength Multichart of up to 10 tickers](#)

**More information:**

[Updated on–line reference](#)

**EQUITY**

Trading system toolbox

**– calculate single–symbol equity line**

(AFL 2.0)

**SYNTAX**     *equity( Flags = 0, RangeType = -1, From = 0, To = 0 )***RETURNS**     ARRAY**FUNCTION**     NOTE: This function is left here for backward compatibility and is using old, single–security backtester. New coding should rather use portfolio–level equity (special ~~~EQUITY ticker).

Function:

Returns single–security Equity line based on buy/sell/short/cover rules, buy/sell/short/coverprice arrays, all apply stops, and all other backtester settings. *Flags* – defines the behaviour of Equity function

- 0** : (default) Equity works as in 3.98 – just calculates the equity array
- 1** : works as 0 but additionally updates buy/sell/short/cover arrays so all redundant signals are removed exactly as it is done internally by the backtester plus all exits by stops are applied so it is now possible to visualise ApplyStop() stops.
- 2** : (advanced) works as 1 but updated signals are not moved back to their original positions if buy/sell/short/cover delays set in preferences are non–zero. Note: this value of flag is documented but in 99% of cases should not be used in your formula. Other values are reserved for the future.

*RangeType* – defines quotations range being used:

- 1** : (default) use range set in the Automatic analysis window
- 0** : all quotes
- 1** : n last quotes (n defined by 'From' parameter)
- 2** : n last days (n defined by 'From' parameter)
- 3** : From/To dates

*From* : defines start date (datenum) (when RangeType == 3) or "n" parameter (when RangeType == 1 or 2)

*To*: defines end date (datenum) (when RangeType == 3) otherwise ignored

datenum defines date the same way as DateNum() function as YYYYMMDD where YYYY is (year – 1900), MM is month, DD is day

December 31st, 1999 has a datenum of 991231

May 21st, 2001 has a datenum of 1010521 All these parameters are evaluated at the time of the call of Equity function. Complete equity array is generated at once. Changes to buy/sell/short/cover rules made after the call have no effect. Equity function can be called multiple times in single formula.

IMPORTANT NOTE: Equity() function uses so called "old" single–security backtester that offers only subset of features of new backtester. To retrieve value of portfolio–level equity generated by new backtester use Foreign("~~~EQUITY", "C").

**EXAMPLE**     *Buy = //your Buy rule;*

```

sell = //your Sell rule;
Graph0 = Equity();

```

**SEE ALSO****Comments:**

<b>Herman van den Bergen</b> psytek@magma.ca 2003-02-23 09:46:19	<p>When the Equity function is called multiple times in a single formula one must be carefull when using it with ApplyStop().</p> <p>Tomasz wrote: "Equity(1) changes buy/sell variables (evaluates stops – and writes them back to buy/sell arrays). If you are using non-zero delays both Equity calls will return different values because in first case exits are generated by stops (not delayed) and in second case STOP signals written back to buy/sell arrays are delayed (opposite to the first case).</p> <p>Equity(1) affects the buy/sell variables. It is not a "no-operation" function. If you want a "no-op" you should use Equity( 0 ) to generate equity line.</p> <p>This is by design and described in the User's Guide. AFL reference: Equity function and chart</p>
<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2003-05-21 17:56:46	<p>Using Equity( 1 ) evaluates stops and writes BACK signals to sell/cover arrays. Equity(1) also removes all extra signals.</p> <p>Depending on kind of the stop various values are written back to sell/cover array to enable you to distinguish if given signal was generated by regular rule or by stop.</p> <p>1 – regular exit  2 – max. loss  3 – profit target  4 – trailing  5 – n-bar stop  6 – ruin stop</p> <p>... your rules...  ApplyStop( stopTypeTrail, stopModePercent, 10, True );  Equity( 1 );  Writelf( sell == 1, "Regular exit",  Writelf( sell == 4, "Trailing stop", "" ) );</p>
<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2003-05-29 05:27:07	<p>When your formula uses Equity(1) you should avoid using built-in delays. Here is a story why:</p> <p>Only BACKTESTER implements delays while EXPOLORATION and other modes do NOT.</p> <p>Therefore Equity(1) must not delay signals by itself. However in order to perform equity calculations delays must be applied to match backtester output, so</p>

	<p>AmiBroker when it encounters Equity(1) applies the delays (even in exploration, indicator, etc) but just before end of the equity call AmiBroker must ADJUST BACK the signals, so Equity-adjusted buy/sell/short/cover arrays do NOT have delay applied.</p> <p>This involves shifting updated bars back and this may cause problem if signal occurs on the very last bar (because it is moved by delay outside the range).</p> <p>To disable this shifting back in exploration (so exploration matches the output of backtester with NON-ZERO delays) you need to use Equity( 2 )</p> <p>On the other hand using Equity(2) in backtest formula causes double delay. (one added by the equity function, second added by the backtester pass).</p> <p>Built-in delays are designed to be used in BACKTESTER ONLY. The intention is as follows: set non zero delays in the settings now SINGLE formula can be used to BACKTEST and to get TODAY SIGNALS for trading for tommorrow (in SCAN) mode.</p> <p>Solution 1: Embbded delays in the AFL code itself: Buy = Ref( Buy, -1 ); Sell = Ref( Sell, -1 );</p> <p>Solution 2: When using Equity AND EXPLORATION Use EQUITY( 2 ) but except of backtest mode if( Status("action") != 5 ) e = Equity( 2 );</p>
--	--

**References:**

The **Equity** function is used in the following formulas in AFL on-line library:

- ['R' Channel](#)
- [AFL-Excel](#)
- [Auto-Optimization Framework](#)
- [Chandelier Exit or Advanced Trailing Stop](#)
- [Ed Seykota's TSP: EMA Crossover System](#)
- [SectorRSI](#)
- [Visualization of stoploses and profit in chart](#)

**More information:**

[Updated on-line reference](#)

**EXP****Math functions****– exponential function**

**SYNTAX**      *exp( NUMBER )*  
                  *exp( ARRAY )*

**RETURNS**    NUMBER,  
                  ARRAY

**FUNCTION**    Calculates **e** raised to the NUMBER or ARRAY power.

**EXAMPLE**

**SEE ALSO**    [The log\(\) function](#)

**References:**

The **EXP** function is used in the following formulas in AFL on–line library:

- ['D/9E H'](#)
- ['D/9E H'](#)
- [Andrews Pitchfork](#)
- [Andrews PitchforkV3.3](#)
- [AR\\_Prediction.afl](#)
- [Auto–Optimization Framework](#)
- [Black Scholes Option Pricing](#)
- [Demand Index](#)
- [Ehlers Fisher Transform](#)
- [Log Time Scale](#)
- [MultiCycle 1.0](#)
- [Option Calls, Puts and days till third friday.](#)
- [Probability Calculator](#)
- [Schiff Lines](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**

[Updated on–line reference](#)

**EXREM****Trading system toolbox****– remove excessive signals**

(AFL 1.5)

**SYNTAX**      **exrem( ARRAY1, ARRAY2 )****RETURNS**      ARRAY

**FUNCTION**      removes excessive signals:  
                      returns 1 on the first occurrence of "true" signal in Array1  
                      then returns 0 until Array2 is true even if there are "true" signals in Array1

**EXAMPLE**      buy = ExRem( buy, sell );  
                      sell = ExRem( sell, buy );

**SEE ALSO****References:**

The **EXREM** function is used in the following formulas in AFL on–line library:

- [ADXbuy](#)
- [AFL Example](#)
- [AFL Example – Enhanced](#)
- [An n bar Reversal Indicator](#)
- [Andrews PitchforkV3.3](#)
- [ATR Study](#)
- [ATR Trading System](#)
- [Auto–Optimization Framework](#)
- [Bollinger band normalization](#)
- [Bull Fear / Bear Fear](#)
- [Chandelier Exit or Advanced Trailing Stop](#)
- [Demand Index](#)
- [DMI Spread Index](#)
- [Ema bands](#)
- [FastStochK FullStochK–D](#)
- [Gann HiLo Indicator and System](#)
- [Hilbert Study](#)
- [Hull Moving Average](#)
- [Peterson](#)
- [Pullback System No. 1](#)
- [RSI Double–Bottom](#)
- [RSIS](#)
- [RUTVOL timing signal with BB Scoring routine](#)
- [Schiff Lines](#)
- [SectorRSI](#)
- [STD\\_STK Multi](#)
- [Stochastic Fast%K and Full](#)
- [StochD\\_StochK Single.afl](#)
- [TD sequential](#)
- [The D\\_oscillator](#)
- [The Three Day Reversal](#)
- [Trading ATR 10–1](#)
- [Trend Continuation Factor](#)



- [Triangular Moving Average new](#)
- [Using From and To dates from Auto Analysis in Code](#)
- [Visualization of stoploses and profit in chart](#)
- [Williams Alligator system](#)
- [Zig Zag Indicator with Valid Entry and Exit Points](#)

**More information:**

[Updated on-line reference](#)

**EXREMSPAN**

Trading system toolbox

**– remove excessive signals spanning given number of bars**

(AFL 2.0)

**SYNTAX**      *exremspan( ARRAY1, numbars )***RETURNS**      ARRAY

**FUNCTION**      Removes excessive signals that span *numbars* bars since initial signal. (In other words first non-zero bar passes through, then all subsequent non-zero bars are ignored (zero is returned) until *numbars* bars have passed since initial signal. From then on a new signal may pass through)

This function makes easy writing exits after a fixed number of bars, for example if you want to sell always 5 bars after the buy you should now use combination of ExRemSpan and Ref(). It will work even if initial buy array has lots of redundant signals:

**EXAMPLE**      Buy = 1;  
Buy = ExRemSpan( Buy, 5 );  
Sell = Ref( Buy, -5 );

**SEE ALSO****References:**

The **EXREMSPAN** function is used in the following formulas in AFL on-line library:

- [ekeko price chart](#)
- [SectorRSI](#)

**More information:**

[Updated on-line reference](#)

**FCLOSE**  
**– close a file****File Input/Output functions**  
(AFL 2.5)**SYNTAX**     ***fclose( filehandle )*****RETURNS**     NOTHING**FUNCTION**     Closes a file.

The filehandle (NUMBER) should be the handle returned by **fopen** function.

**EXAMPLE**

```
fh = fopen( "myfile.txt", "w" );
if( fh )
{
    fputs( "Testing", fh );
    fclose( fh );
}
```

**SEE ALSO**     [fopen\(\)](#) function , [fputs\(\)](#) function , [fgets\(\)](#) function

**References:**

The **fclose** function is used in the following formulas in AFL on–line library:

- [Export Intraday Data](#)
- [TWS auto–export Executions–file parser](#)
- [WLBuildProcess](#)

**More information:**

[Updated on–line reference](#)

**FDELETE****– deletes a file****File Input/Output functions**  
(AFL 2.70)**SYNTAX**     *fdelete( "filename" )***RETURNS**    NUMBER**FUNCTION**   This function deletes a file.

"filename" is path to the file name (relative or full path). If just file name without path is specified then AmiBroker directory is used, returns TRUE if file successfully deleted, FALSE otherwise

**EXAMPLE****SEE ALSO**    [fopen\(\)](#) function , [fclose\(\)](#) function**References:**

The **fdelete** function is used in the following formulas in AFL on–line library:

- [TWS auto–export Executions–file parser](#)

**More information:**[Updated on–line reference](#)

**FEOF****File Input/Output functions****– test for end-of-file**

(AFL 2.5)

**SYNTAX**     *feof( filehandle )***RETURNS**    NUMBER

**FUNCTION**    The feof function returns a nonzero value after the first read operation that attempts to read past the end of the file. It returns 0 if the current position is not end of file. There is no error return value.

*filehandle* is a file handle returned by **fopen** function.

**EXAMPLE**

```
//  
// The following code (commentary) reads  
// all lines from the external file and  
// displays it in commentary window  
  
fh = fopen( "quotes.csv", "r" );  
if( fh )  
{  
    while( ! feof( fh ) )  
    {  
        printf( fgets( fh ) );  
    }  
}  
else  
{  
    printf( "ERROR: file can not be found (does not exist)" );  
}
```

**SEE ALSO**    [fopen\(\)](#) function , [fclose\(\)](#) function , [fgets\(\)](#) function

**References:**

The **feof** function is used in the following formulas in AFL on-line library:

- [Calculate composites for tickers in list files](#)
- [TWS auto-export Executions-file parser](#)
- [WLBuildProcess](#)

**More information:**

[Updated on-line reference](#)

**FFT****Basic price pattern detection****– performs Fast Fourier Transform**

(AFL 2.90)

**SYNTAX**     **FFT( array, len = 0 )****RETURNS**     ARRAY**FUNCTION**     The function performs FFT (Fast Fourier Transform) on last 'len' bars of the array, if len is set to zero, then FFT is performed on entire array. len parameter must be even.

Result:

function returns array which holds FFT bins for first 'len' bars. There are len/2 FFT complex bins returned, where bin is a pair of numbers (complex number): first is real part of the complex number and second number is the imaginary part of the complex number.

```
result = FFT( array, 256 );
```

where:

- 0th bin (result[0] and result[1]) represents DC component,
- 1st bin (result[1 ] and result[2]) represents real and imaginary parts of lowest frequency range and so on upto result[ len – 2 ] and result[ len – 1 ]

remaining elements of the array are set to zero.

IMPORTANT note: input array for FFT must NOT contain any Null values. Use Nz() function to convert Nulls to zeros if you are not sure that input array is free from nulls.

FFT bins are complex numbers and do not represent real amplitude and phase. To obtain amplitude and phase from bins you need to convert inside the formula. The following code snippet does that:

```
ffc = FFT(data,Len);
for( i = 0; i < Len - 1; i = i + 2 )
{
    amp[ i ] = amp[ i + 1 ] = sqrt(ffc[ i ]^ 2 + ffc[ i + 1 ]^2);
    phase[ i ] = phase[ i + 1 ] = atan2( ffc[ i + 1], ffc[ i ] );
}
```

**EXAMPLE**     `SetBarsRequired(100000,100000);`

```
Len = Param("FFT Length", 1024, 64, 10000, 10 );
```

```
Len = Min( Len, BarCount );
```

```
x = BarIndex();
```

```
x1 = x - BarCount + Len;
```

```
input = C;
```

```

a = LastValue( LinRegIntercept( input, Len - 1 ) );
b = LastValue( LinRegSlope( input, Len - 1 ) );

Lr = a + b * x1;

data = input - Lr; // de-trending

ffc = FFT(data,Len);

for( i = 0; i < Len - 1; i = i + 2 )
{
    amp[ i ] = amp[ i + 1 ] = sqrt(ffc[ i ]^ 2 + ffc[ i + 1 ]^2);
    phase[ i ] = phase[ i + 1 ] = atan2( ffc[ i + 1 ], ffc[ i ] );
}

auto = ParamToggle("Auto dominant cycle", "No|Yes", 1 );
sbar = Param( "Which FFT bin", 1, 0, 50 );

skipbin1 = ParamToggle("Skip 1st FFT bin", "No|Yes", 1 );

if( auto )
{
    sbar = int( LastValue(ValueWhen( amp == LastValue(Highest( IIf(
skipbin1 AND x < 4, 0 , amp ) )), x / 2 )) );
}

fv = Status("firstvisiblebar");

thisbar = Ref( int(x/2) == sbar, -fv);
Plot( Ref(amp,-fv),
"amplitude (bin " + Ref( int(x/2), -fv ) + ")", IIf( thisbar,
colorRed, colorBlack ),styleArea);

Plot( IIf( BarCount - BarIndex() < Len, data, Null ) ,
"de-trended input (" + Len + " bars)", colorOrange, styleLeftAxisScale
);
Plot( cos( phase[ sbar * 2 ] + (sbar) * x1 * 2 * 3.1415926 / Len ),
" dominant cycle " + Len/(sbar) + "(" + sbar + " bin) bars",
colorBlue, styleOwnScale );

GraphZOrder=1;
GraphXSpace = 10;

```

**SEE ALSO**    [atan2\(\)](#) function

**References:**

The **FFT** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)



**FGETS****File Input/Output functions****– get a string from a file**

(AFL 2.5)

**SYNTAX**     **fgets( filehandle )****RETURNS**     STRING**FUNCTION**     The **fgets** function reads a string from the input file (defined by *filehandle* argument ) and returns it as a result.

**fgets** reads characters from the current file position to and including the first newline character, or to the end of the file whichever comes first. The newline character, if read, is included in the returned string.

The *filehandle* argument is a number returned by **fopen** function. The file has to be opened with "r" flag (for reading).

**EXAMPLE**

```
//
// The following code (commentary) reads
// all lines from the external file and
// displays it in commentary window

fh = fopen( "quotes.csv", "r" );
if( fh )
{
    while( ! feof( fh ) )
    {
        printf( fgets( fh ) );
    }
}
else
{
    printf("ERROR: file can not be found (does not exist)");
}
```

**SEE ALSO**     **fopen()** function , **fclose()** function**References:**

The **fgets** function is used in the following formulas in AFL on–line library:

- [Calculate composites for tickers in list files](#)
- [TWS auto–export Executions–file parser](#)
- [WLBuidProcess](#)

**More information:**

[Updated on–line reference](#)

**FGETSTATUS****File Input/Output functions****– retrieves file status/properties**

(AFL 2.90)

**SYNTAX** *fgetstatus( filename, what, format = 0 )***RETURNS** NUMBER or STRING**FUNCTION** The function that retrieves file properties/status.

Returns NUMBER or STRING depending on format parameter. If file does not exist it returns Null.

Parameters:

- filename – the name of the file (with or without full path) to query
- what – specifies what file property to retrieve, allowable values
  - 0 – the date/time the file was created
  - ◆ 1 – the date/time the file was last modified
  - ◆ 2 – the date/time the file was last accessed for reading
  - ◆ 3 – the file size in bytes
  - ◆ 4 – attribute byte of the file
- format – specifies return format of date/time values (format specifications are the same as in Now() function): allowed values:
  - ◆ 0 – returns string containing date/time formatted according to system settings
  - ◆ 1 – returns string containing date only formatted according to system settings
  - ◆ 2 – returns string containing time only formatted according to system settings
  - ◆ 3 – returns DATENUM number with date
  - ◆ 4 – returns TIMENUM number with time
  - ◆ 5 – returns DATETIME number with date/time
  - ◆ 6 – returns date DAY (1..31)
  - ◆ 7 – returns date MONTH (1..12)
  - ◆ 8 – returns date YEAR (four digit)
  - ◆ 9 – returns date DAY OF WEEK (1..7, where 1=Sunday, 2=Monday, and so on)
  - ◆ 10 – returns date DAY OF YEAR (1..366)

Note that Windows supports only 2 second resolution of file date/time stamps.

**EXAMPLE**    *// get modification date string of portfolio.afl file*  
               *fgetstatus( "formulas\Equity\portfolio.afl",1,0 );*

**SEE ALSO**    *fclose()* function , *fdelete()* function , *feof()* function , *fgets()* function , *mkdir()* function , *fopen()* function , *fputs()* function , *rmdir()* function

**References:**

The **fgetstatus** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)



**FLIP****Trading system toolbox**  
(AFL 1.5)

–

**SYNTAX**     *flip( ARRAY1, ARRAY2 )***RETURNS**    ARRAY

**FUNCTION**    works as a flip/flop device or "latch" (electronic/electric engineers will know what I mean)  
returns 1 from the first occurrence of "true" signal in Array1  
until a "true" occurs in Array2 which resets the state back to zero  
until next "true" is detected in Array1...

**EXAMPLE**    buy = ExRem( buy, sell );  
buy = **Flip**( buy, sell ); // this essentially reverts the process of ExRem – multiple signals are  
back again

**SEE ALSO****References:**

The **FLIP** function is used in the following formulas in AFL on–line library:

- [Candle Stick Demo](#)
- [RUTVOL timing signal with BB Scoring routine](#)
- [Stops Implementation in AFS](#)

**More information:**

[Updated on–line reference](#)

**FLOOR****Math functions****– floor value**

**SYNTAX**      *floor( NUMBER )*  
                  *floor( ARRAY )*

**RETURNS**    NUMBER,  
                  ARRAY

**FUNCTION**    Calculates the highest integer that is less than NUMBER or ARRAY.

**EXAMPLE**     The function `floor( 18.9 )` returns 18.  
                  The formula `floor( -19.9 )` returns -20.

**SEE ALSO**     [CEIL\(\)](#) function

**References:**

The **FLOOR** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [Bullish Percent Index 2 files combined](#)
- [Bullish Percent Index 2004](#)
- [Color Display.afl](#)
- [Luna Phase](#)
- [Option Calls, Puts and days till third friday.](#)
- [PFChart – High/Low prices Sept2003](#)
- [PFchart with range box sizes](#)
- [PF Chart – Close – April 2004](#)
- [Renko Chart](#)
- [tomy\\_frenchy](#)
- [Triangle exploration using PFChart](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**

[Updated on–line reference](#)

**FMKDIR****File Input/Output functions****– creates (makes) a directory**

(AFL 2.70)

**SYNTAX**     *fmkdir( "dirname" )***RETURNS**    NUMBER**FUNCTION**    Creates (makes) a directory.

"dirname" specifies path of the directory to be created. Please note that this function creates only ONE directory at a time. So if you want to create nested directory tree you have to call it multiple times, for example to create C:\MyDirectory\MySubDirectory folder you have to call it twice:

```
fmkdir( "C:\\MyDirectory" );  
fmkdir( "C:\\MyDirectory\\MySubDirectory" );
```

Note also that it is safe to call it even if directory already exists (then no change to file system is applied)

Returns TRUE if directory successfully created, FALSE otherwise

**EXAMPLE****SEE ALSO****References:**

The **fmkdir** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**FOPEN****– open a file****File Input/Output functions**  
(AFL 2.5)**SYNTAX**      *fopen( filename, mode )***RETURNS**      FILE handle**FUNCTION**      Opens file, returns filehandle (NUMBER).

File handle is non-zero if file opened successfully, zero on failure.

Parameters:

- *filename* – STRING – contains the path to the file name. Please note that single backslash in path must be written in AFL as (double backslash)
- *mode* – STRING – access mode can be "r" – for reading, "w" for writing, "a" for appending (and all other regular C-runtime library modes)

**EXAMPLE**

```
fh = fopen( "myfile.txt", "w" );
if( fh )
{
    fputs( "Testing", fh );
}
else
{
    printf( "Error opening file" );
}
```

**SEE ALSO**      [fclose\(\)](#) function , [fputs\(\)](#) function , [fgets\(\)](#) function**References:**The **fopen** function is used in the following formulas in AFL on-line library:

- [Calculate composites for tickers in list files](#)
- [Export Intraday Data](#)
- [TWS auto-export Executions-file parser](#)
- [WLBuildProcess](#)

**More information:**[Updated on-line reference](#)

## FOREIGN – access foreign security data

Referencing other symbol data  
(AFL 1.5)

**SYNTAX**     *foreign( TICKER, DATAFIELD, fixup = 1)*

**RETURNS**     ARRAY

**FUNCTION**     Allows referencing other (than current) tickers in the AFL formulas. TICKER is a string that holds the symbol of the stock. DATAFIELD defines which array is referenced. Allowable data fields: "open", "close", "high", "low", "volume", "interest".  
The last parameter – fixup – accepts following values

- 0 – the holes are not fixed
- 1 – **default value** – missing data bar OHLC fields are all filled using previous bar  
Close and volume is set to zero.

Note: you can use Foreign/RelStrength without specifying last parameter:

Foreign( "ticker", "field" ), RelStrength( "ticker" ) – then the holes will be fixed.

- 2 – (old pre-4.90 behaviour) – causes filling the holes in the data with previous O, H, L, C, V values

Unless you know what you are doing you should use DEFAULT value of fixup parameter (Fixup=1). If you do not use fixup=1, data holes will have the value of Null that you would need to handle by yourself.

**EXAMPLE**     *// EXAMPLE 1:*  
*// Plotting spread between currently selected symbol and another one*  
**Graph0 = Close - Foreign( "MSFT", "Close" ) ;**

```
// EXAMPLE 2:
// Built-in relative performance chart
_N( TickerList = ParamStr( "Tickers", "^DJI,MSFT,GE" ) );
NumBars = 20;
fvb = Status( "firstvisiblebar" );
Plot( 100 * ( C - C[ fvb ] ) / C[ fvb ], Name(), colorBlue );
for( i = 0; ( symbol = StrExtract( TickerList, i ) ) != ""; i++ )
{
    fc = Foreign( symbol, "C" );

    if( ! IsNull( fc[ 0 ] ) )
    {
        Plot( 100 * ( fc - fc[ fvb ] ) / fc[ fvb ],
              symbol,
              colorLightOrange + ( (2*i) % 15 ),
              styleLine );
    }
}
PlotGrid( 0, colorYellow );
_N( Title = "{{NAME}} - Relative Performance [%]: {{VALUES}}" );
```



**SEE ALSO** [PLOTFOREIGN\(\)](#) function , [SetForeign\(\)](#) function

**Comments:**

<p><b>Tomasz Janeczko</b>  tj --at-- amibroker.com  2003-08-07 20:28:41</p>	<p>Foreign function synchronizes the data file you are referencing with the currently selected symbol.</p> <p>Synchronization makes sure that EACH bar of FOREIGN data matches exactly with each bar of currently selected symbol.</p> <p>So if DateNum() function returns 990503 for given bar then Close array represents the CLOSE of currently selected symbol at May 3, 1999 and Foreign("SYMBOL", "C") represents close of foreign symbol at May 3, 1999 TOO.</p> <p>This is absolutely necessary because otherwise you won't be able to do ANY meaningful operations involving both selected symbol and foreign symbol.</p> <p>This also needed for the display so when you mark the quote with vertical line it will always match the date displayed regardless if you use Foreign or not.</p> <p>Please note that if you have data holes in currently selected symbol then in order to synchronize bars Foreign function will remove bars that exist in Foreign symbol but do not exist in currently selected symbol.</p>
---	--

**References:**

The **FOREIGN** function is used in the following formulas in AFL on-line library:

- [30 Week Hi Indicator – Display](#)
- [52 Week New High–New Low Index](#)
- [AccuTrack](#)
- [Alpha and Beta and R\\_Squared Indicator](#)
- [Auto–Optimization Framework](#)
- [Baseline Relative Performance Watchlist charts V2](#)
- [BMTRIX Intermediate Term Market Trend Indicator](#)
- [Bullish Percent Index 2004](#)
- [Dave Landry PullBack Scan](#)
- [Elder Triple Screen Trading System.](#)
- [Index of 30 Wk Highs Vs Lows](#)
- [Indicator Explorer \(ZigZag\)](#)
- [pattern correlation](#)
- [Performance Overview](#)
- [Ranking and sorting stocks](#)
- [Ranking Ticker WatchList](#)
- [Relative Strength Multichart of up to 10 tickers](#)
- [RUTVOL timing signal with BB Scoring routine](#)
- [Sector Tracking](#)
- [SectorRSI](#)
- [The D\\_oscillator](#)

- [Weighted Index](#)
- [WLBuildProcess](#)

**More information:**

[Updated on-line reference](#)

**FPUTS****File Input/Output functions****– write a string to a file**

(AFL 2.5)

**SYNTAX**     *fputs( string, filehandle )***RETURNS**    NOTHING**FUNCTION**    Writes (puts) the *string* to the file.

The *filehandle* must be a number returned by **fopen** function used to open the file. The file has to be open for writing or appending ("w" or "a") for this fputs to work.

**EXAMPLE**

```
//
// The following code exports quotes
// of current stock to quotes.csv
// comma separated file
//

fh = fopen( "quotes.csv", "w" );
if( fh )
{
    fputs( "Date,Open,High,Low,Close,Volume\n", fh );

    y = Year();
    m = Month();
    d = Day();

    for( i = 0; i < BarCount; i++ )
    {
        ds = StrFormat( "%02.0f-%02.0f-%02.0f,",
                        y[ i ], m[ i ], d[ i ] );
        fputs( ds, fh );

        qs = StrFormat( "%.4f, %.4f, %.4f, %.4f, %.0f\n",
                        O[ i ], H[ i ], L[ i ], C[ i ], V[ i ] );
        fputs( qs, fh );
    }

    fclose( fh );
}
```

**SEE ALSO**    [fopen\(\)](#) function , [fclose\(\)](#) function , [fgets\(\)](#) function**References:**

The **fputs** function is used in the following formulas in AFL on–line library:

- [Export Intraday Data](#)
- [TWS auto–export Executions–file parser](#)

**More information:**

[Updated on-line reference](#)

**FRAC****Math functions****– fractional part**

**SYNTAX**      *frac( NUMBER )*  
                 *frac( ARRAY )*

**RETURNS**    NUMBER,  
                 ARRAY

**FUNCTION**    Eliminates the integer portion of NUMBER or ARRAY and returns the fractional part.

**EXAMPLE**     The formula `frac( 12.4 )` returns 0.4; the formula `frac(–15.7 )` returns –0.7.

**SEE ALSO**     [The int\(\) function](#)

**References:**

The **FRAC** function is used in the following formulas in AFL on–line library:

- [Bullish Percent Index 2 files combined](#)
- [Cycle Highlighter](#)
- [Cycle Highlighter \(auto best–fit\)](#)
- [Kagi Chart](#)
- [LunarPhase](#)
- [Steve Woods' Cumulative Vol. Percentage Indicator](#)
- [Triangle exploration using PFChart](#)

**More information:**

[Updated on–line reference](#)

**FRMDIR****File Input/Output functions****– removes a directory**

(AFL 2.70)

**SYNTAX**     *frmdir("dirname")***RETURNS**    NUMBER**FUNCTION**   This function removes a directory

"dirname" specifies path of the directory to be removed. Please note that this function removes only ONE directory at a time. So if you want to remove nested directory tree you have to call it multiple times, for example:

```
frmdir( "C:\\MyDirectory\\MySubDirectory" ); // delete nested subdir  
first  
frmdir( "C:\\MyDirectory" );
```

Note that directory must be empty before removing it otherwise it will not be possible to remove it.

Returns TRUE if directory successfully removed, FALSE otherwise

**EXAMPLE****SEE ALSO****References:**

The **frmdir** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**FULLNAME****Information / Categories****– full name of the symbol**

(AFL 1.1)

**SYNTAX**     *FullName()***RETURNS**     STRING**FUNCTION**     The function returns stock full name which is definable by the user in **Symbol | Information** window.**EXAMPLE**     `printf( fullname() );`**SEE ALSO****References:**

The **FullName** function is used in the following formulas in AFL on–line library:

- [AFL Example](#)
- [AFL Example – Enhanced](#)
- [Bottom Trader](#)
- [colored CCI](#)
- [Commodity Channel Index](#)
- [Dave Landry PullBack Scan](#)
- [Dinapoli Guru Commentary](#)
- [Double Smoothed Stochastic from W.Bressert](#)
- [DPO with shading](#)
- [Elder Triple Screen Trading System.](#)
- [FastStochK FullStochK–D](#)
- [Gordon Rose](#)
- [hassan](#)
- [MACD commentary](#)
- [MACD Histogram – Change in Direction](#)
- [Performance Overview](#)
- [PF Chart – Close – April 2004](#)
- [Pivot Finder](#)
- [prakash](#)
- [Relative Strength](#)
- [Steve Woods' Cum. Vol. Float + Cum. Vol. Channels](#)
- [Steve Woods' Cumulative Vol. Percentage Indicator](#)
- [Steve Woods' Float Channel Lines](#)
- [Stochastic Fast%K and Full](#)
- [Triangular Moving Average new](#)
- [Woodies CCI](#)

**More information:**

[Updated on–line reference](#)

**GAPDOWN**  
**– gap down****Basic price pattern detection****SYNTAX**     *gapdown()***RETURNS**     ARRAY**FUNCTION**     Gives a "1" or "true" on the day a security's prices gap down. Otherwise the result is "0". A gap down occurs if yesterday's low is greater than today's high.**EXAMPLE****SEE ALSO****References:**

The **GAPDOWN** function is used in the following formulas in AFL on–line library:

- [Candle Identification](#)
- [Candle Pattern Function](#)
- [Candle Stick Analysis](#)
- [Candlestick Commentary](#)
- [Candlestick Commentary Modified](#)
- [Candlestick Commentary–modified](#)

**More information:**

[Updated on–line reference](#)



**GAPUP**  
**– gap up****Basic price pattern detection****SYNTAX**     *gapup()***RETURNS**     ARRAY**FUNCTION**     Gives a "1" or "true" on the day a security's prices gap up. Otherwise the result is "0". A gap up occurs if yesterday's high is less than today's low.**EXAMPLE****SEE ALSO****References:**

The **GAPUP** function is used in the following formulas in AFL on–line library:

- [Candle Identification](#)
- [Candle Pattern Function](#)
- [Candle Stick Analysis](#)
- [Candlestick Commentary](#)
- [Candlestick Commentary Modified](#)
- [Candlestick Commentary–modified](#)

**More information:**

[Updated on–line reference](#)

**GETBASEINDEX****– retrieves symbol of relative strength base index****Referencing other symbol data**  
(AFL 2.1)**SYNTAX**     *GetBaseIndex( )***RETURNS**     STRING**FUNCTION**     Retrieves base relative–strength index for given security as defined in Symbol→Categories.**EXAMPLE**     AddTextColumn( GetBaseIndex(), "Base index" );**SEE ALSO****References:**

The **GETBASEINDEX** function is used in the following formulas in AFL on–line library:

- [Dave Landry PullBack Scan](#)
- [Elder Triple Screen Trading System.](#)
- [Indicator Explorer \(ZigZag\)](#)

**More information:**

[Updated on–line reference](#)

**GETCATEGORYSYMBOLS**

– retrieves comma-separated list of symbols belonging to given category

Information /  
Categories  
(AFL 2.4)

**SYNTAX**      *GetCategorySymbols( category, index )*

**RETURNS**    STRING

**FUNCTION**    IMPORTANT: This function is now available under new name of **CategoryGetSymbols**. The old name is left only for backward compatibility. Please use new name in all new codes.

**EXAMPLE**

**SEE ALSO**    [CategoryGetSymbols\(\)](#) function

**References:**

The **GetCategorySymbols** function is used in the following formulas in AFL on-line library:

- [Baseline Relative Performance Watchlist charts V2](#)
- [Count Tickers in Watchlist](#)
- [Ranking and sorting stocks](#)
- [Ranking Ticker WatchList](#)
- [WLBuildProcess](#)

**More information:**

[Updated on-line reference](#)

**GETCHARTID****Exploration / Indicators****– get current chart ID**

(AFL 2.3)

**SYNTAX**     *GetChartID()***RETURNS**    NUMBER**FUNCTION**    returns the chart ID of current indicator formula. Returns 0 if used in Automatic analysis.**EXAMPLE**     Cross( graph0, Study( "RE", GetChartID() ) );**SEE ALSO****References:**

The **GETCHARTID** function is used in the following formulas in AFL on–line library:

**More information:**[Updated on–line reference](#)

## GETCURSORMOUSEBUTTONS

– get current state of mouse buttons

**Indicators**  
(AFL 2.80)

**SYNTAX**     *GetCursorMouseButtons()*

**RETURNS**    NUMBER

**FUNCTION**    This function returns mouse button state at the time when chart formula is executed.

- 0 – if no mouse button is pressed
- 1 – if left mouse button is pressed
- 2 – if right mouse button is pressed
- 4 – if middle mouse button is pressed

plus combinations:

- 3 – left + right
- 5 – left + middle
- 6 – right + middle
- 7 – left + right + middle

**EXAMPLE**

```
if( GetCursorMouseButtons() & 1 )
{
    printf( "left mouse button is pressed down" );
}
```

**SEE ALSO**    [GetCursorXPosition\(\)](#) function , [GetCursorYPosition\(\)](#) function

### References:

The **GetCursorMouseButtons** function is used in the following formulas in AFL on–line library:

### More information:

[Updated on–line reference](#)

**GETCURSORXPOSITION****Indicators**  
(AFL 2.80)**– get current X position of mouse pointer****SYNTAX**     *GetCursorXPosition()***RETURNS**     NUMBER (datetime)**FUNCTION**     Retrieves current mouse pointer X co-ordinate (i.e. datetime of bar under the mouse pointer).

Values returned are equal to those visible in the status bar, and these functions require status bar to be visible. Returned values represent cursor position at the formula execution time (or few milliseconds before it) and accuracy is subject to pixel resolution of the screen (first cursor position is read in screen pixels (integer) and then converted to actual value therefore for example when screen resolution is 1024x768 maximum obtainable resolution in X direction is 0.1% and in Y direction 0.13%), also X values are snap to datetime of nearest data bar.

It only makes sense to use these functions in indicator/interpretation code.

Using them in AA window may yield random values. *GetCursorXPosition()* function returns X position in DateTime format (the same as used by *DateTime()* function). You can convert it to string using *DateTimeToStr()* function. *GetCursorYPosition()* returns Y position (as displayed in Y axis of the chart).

**EXAMPLE**     `ToolTip = "X=" + DateTimeToStr( GetCursorXPosition() ) +  
                  "\nY=" + GetCursorYPosition();`

**SEE ALSO**     *GetCursorYPosition()* function , *GetCursorMouseButtons()* function

**References:**

The **GetCursorXPosition** function is used in the following formulas in AFL on-line library:

- [Visualization of stoploses and profit in chart](#)

**More information:**

[Updated on-line reference](#)

**GETCORSORYPOSITION****– get current Y position of mouse pointer****Indicators**  
(AFL 2.80)**SYNTAX**     *GetCursorYPosition()***RETURNS**    NUMBER**FUNCTION**    Retrieves current mouse pointer Y co-ordinate (i.e. value in dollars or other Y-axis unit).

Values returned are equal to those visible in the status bar, and these functions require status bar to be visible. Returned values represent cursor position at the formula execution time (or few milliseconds before it) and accuracy is subject to pixel resolution of the screen (first cursor position is read in screen pixels (integer) and then converted to actual value therefore for example when screen resolution is 1024x768 maximum obtainable resolution in X direction is 0.1% and in Y direction 0.13%), also X values are snap to datetime of nearest data bar.

It only makes sense to use these functions in indicator/interpretation code.

Using them in AA window may yield random values. *GetCursorXPosition()* function returns X position in DateTime format (the same as used by *DateTime()* function). You can convert it to string using *DateTimeToStr()* function. *GetCursorYPosition()* returns Y position (as displayed in Y axis of the chart).

**EXAMPLE**     `ToolTip = "X=" + DateTimeToStr( GetCursorXPosition() ) +  
"nY=" + GetCursorYPosition();`

**SEE ALSO**     *GetCursorXPosition()* function , *GetCursorMouseButtons()* function

**References:**

The **GetCursorYPosition** function is used in the following formulas in AFL on-line library:

- [Visualization of stoploses and profit in chart](#)

**More information:**

[Updated on-line reference](#)

## GETDATABASENAME

Information / Categories

– retrieves folder name of current database

(AFL 2.3)

**SYNTAX**      *GetDatabaseName()*

**RETURNS**     STRING

**FUNCTION**    retrieves the name of the database – the last part (folder) of the database path

**EXAMPLE**

**SEE ALSO**

**References:**

The **GetDatabaseName** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)



**GETEXTRADATA****– get extra data from external data source****Miscellaneous functions**  
(AFL 1.9)**SYNTAX**     *GetExtraData(***RETURNS**    NUMBER or ARRAY or STRING**FUNCTION**    Retrieves data–source specific data.  
Currently only Quotes Plus and TC2000 plug–in support this function.  
The list of fields available via QP2 plug–in:

- "AnnDividend"
- "Shares"
- "SharesFloat"
- "IssueType" (string)
- "SharesOut"
- "SharesShort"
- "TTMsales"
- "Beta"
- "TTMEps"
- "HiPERange"
- "LoPERange"
- "PEG"
- "InstHolds"
- "LTDebtToEq"
- "CashFlowPerShare"
- "ROE"
- "TMSales"
- "Yr1EPSGrowth"
- "Yr5EPSGrowth"
- "Yr1ProjEPSGrowth"
- "Yr2ProjEPSGrowth"
- "Yr3to5ProjEPSGrowth"
- "BookValuePerShare"
- "Briefing" (string)
- "QRS" (array)
- "HasOptions"
- "EPSRank" (array) – requires QP plugin 1.4.3
- "Sales" (array) – requires QP plugin 1.5.0
- "EPS" (array) – requires QP plugin 1.5.0
- "LastMainDate" – (number) date of last update of given symbol in YYYYMMDD format – requires QP plugin 1.5.1
- "Exchange" – (string) – exchange code – requires QP plugin 1.5.1
- "ExchangeSub" – (string) exchange sub–code – requires QP plugin 1.5.1
- "Flags" – (string) – requires QP plugin 1.5.1
- "MarginFlag" – (string)– requires QP plugin 1.5.1
- "CUSIP" – (string) – requires QP plugin 1.5.1
- "SIC" – (string) – requires QP plugin 1.5.1
- "IssueStatus" – (string) – with default settings all symbols should have issue status = 0 other possible values: 0 = actively trading; 1, P = trading on a when issued basis, 5, 6, 7, A, B, C, D, E, M = not trading 4, N = new symbol, G, K, X, R, Z = changes to

symbol, cusip, name, etc. – requires QP plugin 1.5.1

The list of fields available via TC2000 plug-in:

- "BOP" – balance of power indicator
- "MoneyStream" – money stream indicator

**EXAMPLE**     `GetExtraData("briefing"); /* gives briefing text (STRING) */`  
                 `graph0 = GetExtraData("QRS"); /*gives Quotes Plus relative strength (ARRAY) */`

**SEE ALSO**

**References:**

The **GETEXTRADATA** function is used in the following formulas in AFL on-line library:

- [Steve Woods' Cum. Vol. Float + Cum. Vol. Channels](#)
- [Steve Woods' Cumulative Vol. Percentage Indicator](#)
- [Steve Woods' Float Channel Lines](#)

**More information:**

[Updated on-line reference](#)

**GETFNDATA****– get fundamental data****Information / Categories**

(AFL 2.90)

**SYNTAX**     *GetFnData("field")***RETURNS**    NUMBER**FUNCTION**    GetFnData allows accessing fundamental data from Information window (View->Information)  
"field" parameter can be one of the following:

- "EPS"
- "EPSEstCurrentYear"
- "EPSEstNextYear"
- "EPSEstNextQuarter"
- "PEGRatio"
- "SharesFloat"
- "SharesOut"
- "DividendPayDate"
- "ExDividendDate"
- "BookValuePerShare"
- "DividendPerShare"
- "ProfitMargin"
- "OperatingMargin"
- "OneYearTargetPrice"
- "ReturnOnAssets"
- "ReturnOnEquity"
- "QtrlyRevenueGrowth"
- "GrossProfitPerShare"
- "SalesPerShare"
- "EBITDAPerShare"
- "QtrlyEarningsGrowth"
- "InsiderHoldPercent"
- "InstitutionHoldPercent"
- "SharesShort"
- "SharesShortPrevMonth"
- "ForwardDividendPerShare"
- "ForwardEPS"
- "OperatingCashFlow"
- "LeveredFreeCashFlow"
- "Beta"
- "LastSplitRatio"
- "LastSplitDate"

**EXAMPLE**     `AddColumn( Close / GetFnData( "EPS" ) , "Current P/E ratio" );`  
                  `AddColumn( Close / GetFnData( "EPSEstNextYear" ) , "Est. Next Year`  
                  `P/E ratio" );`  
                  `Filter = Status( "lastbarinrange" );`

**SEE ALSO**     `GetRTData()` function , `GetRTDataForeign()` function

**References:**

The **GetFnData** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)

**GETOPTION**

Trading system toolbox

**– gets the value of option in automatic analysis settings**

(AFL 2.60)

**SYNTAX**     *GetOption("fieldname")***RETURNS**    NUMBER**FUNCTION**   Gets the value of various options in automatic analysis settings.*field* – is a string that defines the option to read. There are following options available:

- "NoDefaultColumns" – if set to True – exploration does not have default Ticker and Date/Time columns
- "InitialEquity"
- "AllowSameBarExit"
- "ActivateStopsImmediately"
- "AllowPositionShrinking"
- "FuturesMode"
- "InterestRate"
- "MaxOpenPositions" – maximum number of simultaneously open positions (trades) in portfolio backtest/optimization
- "WorstRankHeld" – the worst rank of symbol to be held in rotational trading mode (see **EnableRotationalTrading** for more details)
- "MinShares" – the minimum number of shares required to open the position in the backtester/optimizer. If you don't have enough funds to purchase that many, trade will NOT be entered
- "MinPosValue" – (4.70.3 and above) the minimum dollar amount required to open the position in the backtester/optimizer. If you don't have enough funds trade will NOT be entered
- "PriceBoundChecking" – if set to False – disables checking and adjusting buyprice/sellprice/coverprice/shortprice arrays to current symbol High–Low range.
- CommissionMode –
  - 0 – use portfolio manager commission table
  - 1 – percent of trade
  - 2 – \$ per trade
  - 3 – \$ per share/contract
- CommissionAmount – amount of commission in modes 1..3
- AccountMargin – (in old versions it was 'MarginRequirement') – account margin requirement (as in settings), 100 = no margin
- ReverseSignalForcesExit – reverse entry signal forces exit of existing trade (default = True )
- UsePrevBarEquityForPosSizing – Affects how percent of current equity position sizing is performed.  
False (default value) means: use current (intraday) equity to perform position sizing,  
True means: use previous bar closing equity to perform position sizing

**EXAMPLE**     `InitialEquity = GetOption("InitialEquity");`**SEE ALSO**     [SetOption\(\)](#) function

**References:**

The **GetOption** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)

**GETPERFORMANCECOUNTER**

– retrieves the current value of the high–resolution performance counter

Miscellaneous  
functions  
(AFL 2.90)

**SYNTAX**      *GetPerformanceCounter( bReset = False )*

**RETURNS**    NUMBER

**FUNCTION**    GetPerformanceCounter retrieves the current value of the high–resolution performance counter. Returned value is in milliseconds. Resolution is upto 0.001 ms (1 microsecond). The value of high–resolution counter represents number of milliseconds from either system start (boot) or from last counter reset. To reset the counter you need to call GetPerformanceCounter function with bReset parameter set to True.

Note that resetting counters inside one formula does not affect counters in other formulas. Since returned values are very large (time in milliseconds since system start is usually quite large), for precise measurements of single function or small function block execution times it is strongly recommended to reset counter at the beginning of the block so floating point resolution (7 digits) does not affect the precision of measurement.

GetPerformanceCounter function can be also used in trading system automation to measure time in milliseconds between various events (just subtract values returned by GetPerformanceCounter() during two different events)

Caveat: this function relies on Windows API QueryPerformanceCounter function and CPU RTDSC instruction and it may yield to inaccurate results if you have multiple–core processor and AMD's "Cool and Quiet" enabled in BIOS or other CPU clock stepping technologies enabled. If this applies to you, you may check Microsoft hotfix to this problem at: <http://support.microsoft.com/?id=896256>

**EXAMPLE**

```

////////////////////////////////////
// EXAMPLE 1
// The code shows that 1000 iterations of sin() calculation
// takes about 1.7 milliseconds.
// Note that call to the GetPerformanceCounter()
// has overhead of about 0.015 ms (15 microseconds)

GetPerformanceCounter(True); // reset counter to zero
for( i = 0; i < 1000; i++ )
{
    k = sin( i );
}

elapsed=GetPerformanceCounter();

"Time [ms] = "+elapsed;

////////////////////////////////////

```

```
// EXAMPLE 2
// GetPerformanceCounter function
// may also be used to report time since system start.

elapsed=GetPerformanceCounter();

StrFormat("Time since system start %.0f days, %.0f hours, %.0f
minutes, %.0f seconds, %.0f milliseconds ",
floor(elapsed/(24*60*60*1000)),
floor( elapsed/(60*60*1000) ) % 24,
floor( elapsed/(60*1000) ) % 60,
floor( elapsed/1000 ) % 60,
elapsed % 1000 );
```

**SEE ALSO****References:**

The **GetPerformanceCounter** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)



## GETPLAYBACKDATETIME

– get bar replay position date/time

**Date/Time**  
(AFL 3.0)

**SYNTAX**     *GetPlaybackDateTime()*

**RETURNS**    NUMBER

**FUNCTION**    The function returns bar replay playback position in datetime format, or zero if bar replay is NOT active

**EXAMPLE**

```
pt = GetPlaybackDateTime(); // new function to retrieve playback
position date/time,
//returns zero if bar replay is NOT active

if( pt )
{
    Title = "Playback time: " + DateTimeToStr( pt );
}
else
{
    Title = "Bar Replay not active";
}
```

**SEE ALSO**    [DateTime\(\)](#) function , [DateTimeConvert\(\)](#) function , [DateTimeToStr\(\)](#) function

### References:

The **GetPlaybackDateTime** function is used in the following formulas in AFL on–line library:

### More information:

[Updated on–line reference](#)

## GETPRICESTYLE

### – get current price chart style

Exploration / Indicators  
(AFL 2.70)

**SYNTAX**      *GetPriceStyle*

**RETURNS**    NUMBER

**FUNCTION**   Returns price chart style value to be used in Plot statement Returned value depends on selection in View→Price chart style menu

**EXAMPLE**    `Plot( C, "Close", ParamColor("Color", colorBlack ), styleNoTitle |  
ParamStyle("Style") | GetPriceStyle() );`

#### SEE ALSO

#### References:

The **GetPriceStyle** function is used in the following formulas in AFL on–line library:

- [Dave Landry PullBack Scan](#)
- [Elder Triple Screen Trading System.](#)
- [Fibonacci Moving averages](#)
- [LunarPhase](#)
- [nifty](#)
- [prakash](#)
- [SAR–ForNextBarStop](#)

#### More information:

[Updated on–line reference](#)

**GETRTDATA****– retrieves the real-time data fields****Miscellaneous functions**  
(AFL 2.60)**SYNTAX**     *GetRTData("fieldname")***RETURNS**    NUMBER**FUNCTION**    Retrieves the LAST (the most recent) value of the following fields reported by streaming real time data source:

- "Ask" – current best ask price
- "AskSize " – current ask size
- "Bid" – current best bid price
- "BidSize " – current bid size
- "52WeekHigh" – 52 week high value
- "52WeekHighDate" – 52 week high date (in datenum format)
- "52WeekLow" – 52 week low value
- "52WeekLowDate" – 52 week low date (in datenum format)
- "Change" – change since yesterdays close
- "Dividend" – last dividend value
- "DivYield" – dividend yield
- "EPS" – earnings per share
- "High" – current day's high price
- "Low" – current day's low price
- "Open" – current day's open price
- "Last" – last trade price
- "OpenInt" – current open interest
- "Prev" – previous day close
- "TotalVolume" – total today's volume
- "TradeVolume" – last trade volume
- "ChangeDate" – datenum (YYYYMMDD) of last data change
- "ChangeTime" – timenum (HHMMSS) of last data change
- "UpdateDate" – datenum (YYYYMMDD) of last data update
- "UpdateTime" – timenum (HHMMSS) of last data update
- "Shares" – total number of shares

Note 1: this function is available ONLY in PROFESSIONAL edition, calling it using Standard edition will give you NULL values for all fields

Note 2: works only if data source uses real time data source (plugin)

Note 3: availability of data depends on underlying data source – check the real-time quote window to see if given field is available

Note 4: function result represents the current value at the time of the call /formula execution/, and they will be refreshed depending on chart or commentary refresh interval /settable in preferences/. Built-in real time quote window is refreshed way more often (at least 10 times per second)

**EXAMPLE**     `"Bid" = "+GetRTData( "Bid" ) ;`  
                   `"Ask" = "+GetRTData( "Ask" ) ;`

```
"Last" = "+GetRTData( "Last" );  
"Vol" = "+GetRTData( "TradeVolume" );  
  
"EPS" = "+GetRTData( "EPS" );  
"52week high" = "+GetRTData( "52weekhigh" );
```

**SEE ALSO** [GetRTDataForeign\(\)](#) function

#### References:

The **GetRTData** function is used in the following formulas in AFL on-line library:

#### More information:

[Updated on-line reference](#)

**GETRTDATAFOREIGN****Miscellaneous functions****– retrieves the real-time data fields (for specified symbol)**

(AFL 2.80)

**SYNTAX**      *GetRTDataForeign( "fieldname" , "symbol" )***RETURNS**      NUMBER**FUNCTION**      This function is similar to GetRTData but allows to specify symbol OTHER than currently selected and it is much faster than SetForeign/GetRTData combo.

The function retrieves the LAST (the most recent) value of the following fields reported by streaming real time data source for specified symbol:

- "Ask" – current best ask price
- "AskSize " – current ask size
- "Bid" – current best bid price
- "BidSize " – current bid size
- "52WeekHigh" – 52 week high value
- "52WeekHighDate" – 52 week high date (in datenum format)
- "52WeekLow" – 52 week low value
- "52WeekLowDate" – 52 week low date (in datenum format)
- "Change" – change since yesterdays close
- "Dividend" – last dividend value
- "DivYield" – dividend yield
- "EPS" – earnings per share
- "High" – current day's high price
- "Low" – current day's low price
- "Open" – current day's open price
- "Last" – last trade price
- "OpenInt" – current open interest
- "Prev" – previous day close
- "TotalVolume" – total today's volume
- "TradeVolume" – last trade volume
- "ChangeDate" – datenum (YYYYMMDD) of last data change
- "ChangeTime" – timenum (HHMMSS) of last data change
- "UpdateDate" – datenum (YYYYMMDD) of last data update
- "UpdateTime" – timenum (HHMMSS) of last data update
- "Shares" – total number of shares

Note 1: this function is available ONLY in PROFESSIONAL edition, calling it using Standard edition will give you NULL values for all fields

Note 2: works only if data source uses real time data source (plugin)

Note 3: availability of data depends on underlying data source – check the real-time quote window to see if given field is available

Note 4: function result represents the current value at the time of the call /formula execution/, and they will be refreshed depending on chart or commentary refresh interval /settable in preferences/. Built-in real time quote window is refreshed way more often (at least 10 times per second)

**EXAMPLE**    `"Bid = "+GetRTDataForeign( "Bid" );`  
                 `"Ask = "+GetRTData( "Ask" );`  
                 `"Last = "+GetRTData( "Last" );`  
                 `"Vol = "+GetRTData( "TradeVolume" );`  
  
                 `"EPS = "+GetRTDataForeign( "EPS", "AAPL" );`  
                 `"52week high = "+GetRTDataForeign( "52weekhigh", "MSFT" );`

**SEE ALSO**    `GetRTData()` function , `SetForeign()` function

**References:**

The **GetRTDataForeign** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**GETSCRIPTOBJECT****– get access to script COM object****Miscellaneous functions**  
(AFL 1.8)**SYNTAX**     *getscriptobject( )***RETURNS**    OBJECT**FUNCTION**    Retrieves AFL host's script object. This allows to call functions defined in JScript/VBScript directly from AFL.

**EXAMPLE**    EnableScript("jscript")  
                  <%  
                  function MyAdd(x, y)  
                  {  
                  return x+y;  
                  }  
                  %>  
                  script = GetScriptObject();  
                  WriteVal( script.MyAdd( 7, 9 ) ); // call the function defined in JScript

**SEE ALSO****References:**

The **GETSCRIPTOBJECT** function is used in the following formulas in AFL on–line library:

- [accum/dist mov avg crossover SAR](#)
- [AR\\_Prediction.afl](#)
- [nth \( 1 – 8 \) Order Polynomial Fit](#)
- [Polarized Fractal Efficiency](#)
- [Standard Error Bands](#)
- [tomy\\_frenchy](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**

[Updated on–line reference](#)

**GETTRADINGINTERFACE****– retrieves OLE automation object to automatic trading interfac****Trading system  
toolbox  
(AFL 2.70)****SYNTAX**     *GetTradingInterface(***RETURNS**    OBJECT

**FUNCTION**    Retrieves OLE automation object to automatic trading interface. "Name" is the interface name. You have to have trading interface installed separately to make it work otherwise you will get the error message attempting to use this function. Trading interface for Interactive Brokers is available from download section: <http://www.amibroker.com/download.html>

**EXAMPLE****SEE ALSO****References:**

The **GetTradingInterface** function is used in the following formulas in AFL on–line library:

- [Moving Averages NoX](#)

**More information:**

[Updated on–line reference](#)



## GFXARC

### – draw an arc

**Low-level graphics**  
(AFL 3.0)

**SYNTAX** `GfxArc( x1, y1, x2, y2, x3, y3, x4, y4 )`

**RETURNS** NOTHING

**FUNCTION** Draws an elliptical arc. The arc drawn by using the function is a segment of the ellipse defined by the specified bounding rectangle.

The actual starting point of the arc is the point at which a ray drawn from the center of the bounding rectangle through the specified starting point intersects the ellipse. The actual ending point of the arc is the point at which a ray drawn from the center of the bounding rectangle through the specified ending point intersects the ellipse. The arc is drawn in a counterclockwise direction.

Parameters

- x1 – x-coordinate of the upper left corner of the bounding rectangle
- y1 – y-coordinate of the upper left corner of the bounding rectangle
- x2 – x-coordinate of the lower right corner of the bounding rectangle
- y2 – y-coordinate of the lower right corner of the bounding rectangle
- x3 – x-coordinate of the arc's starting point.
- y3 – y-coordinate of the arc's starting point.
- x4 – x-coordinate of the arc's ending point.
- y4 – y-coordinate of the arc's ending point.

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

#### EXAMPLE

```
GfxSelectPen( colorRed );
GfxArc( 100, 0, 200, 100, 150, 0, 200, 50 );
```

**SEE ALSO** [GfxChord\(\)](#) function , [GfxPie\(\)](#) function , [GfxSelectPen\(\)](#) function

#### References:

The **GfxArc** function is used in the following formulas in AFL on-line library:

#### More information:

[Updated on-line reference](#)

**GFXCHORD**  
– draw a chord**Low-level graphics**  
(AFL 3.0)**SYNTAX**     *GfxChord( x1, y1, x2, y2, x3, y3, x4, y4 )***RETURNS**    NOTHING

**FUNCTION**    Draws a chord (a closed figure bounded by the intersection of an ellipse and a line segment). The (x1, y1) and (x2, y2) parameters specify the upper-left and lower-right corners, respectively, of a rectangle bounding the ellipse that is part of the chord. The (x3, y3) and (x4, y4) parameters specify the endpoints of a line that intersects the ellipse. The chord is drawn by using the selected pen and filled by using the selected brush.

## Parameters

- x1 – x-coordinate of the upper left corner of the bounding rectangle
- y1 – y-coordinate of the upper left corner of the bounding rectangle
- x2 – x-coordinate of the lower right corner of the bounding rectangle
- y2 – y-coordinate of the lower right corner of the bounding rectangle
- x3 – x-coordinate of the chord's starting point.
- y3 – y-coordinate of the chord's starting point.
- x4 – x-coordinate of the chord's ending point.
- y4 – y-coordinate of the chord's ending point.

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**     *GfxSelectPen( colorRed );*  
                  *GfxSelectSolidBrush( colorBlue );*  
                  *GfxChord( 100, 0, 200, 100, 150, 0, 200, 50 );*

**SEE ALSO**     *GfxSelectPen()* function , *GfxSelectSolidBrush()* function , *GfxPie()* function

**References:**

The **GfxChord** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)

**GFXCIRCLE**  
**– draw a circle****Low-level graphics**  
(AFL 3.0)**SYNTAX**     *GfxCircle( x, y, radius )***RETURNS**     NOTHING**FUNCTION**     Draws a circle. The center of the circle is given by x and y parameters. The circle is drawn with the current pen, and its interior is filled with the current brush.

Parameters

- x – x-coordinate of the center of the circle
- y – y-coordinate of the the center of the circle
- radius – radius of the circle

This function is essentially the same as GfxEllipse( x – radius, y – radius, x + radius, y + radius );

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**     GfxCircle( 100, 100, 50 );**SEE ALSO**     [GfxEllipse\(\)](#) function , [GfxRoundRect\(\)](#) function , [GfxSelectPen\(\)](#) function , [GfxSelectSolidBrush\(\)](#) function**References:**

The **GfxCircle** function is used in the following formulas in AFL on-line library:

**More information:**[Updated on-line reference](#)

**GFXDRAWTEXT****Low-level graphics**  
(AFL 3.0)**– draw a text (clipped to rectangle)****SYNTAX**     *GfxDrawText( "text", left, top, right, bottom, format = 0 )***RETURNS**    NOTHING**FUNCTION**    Formats and draws text in the given rectangle. It formats text by expanding tabs into appropriate spaces, aligning text to the left, right, or center of the given rectangle, and breaking text into lines that fit within the given rectangle. The type of formatting is specified by format argument. When format is not specified the text is aligned to the top/left corner.

Parameters:

- **"text"** – string to be drawn
- **left** – x-coordinate of upper left corner of the clipping rectangle
- **top** – y-coordinate of upper left corner of the clipping rectangle
- **right** – x-coordinate of lower right corner of the clipping rectangle
- **bottom** – y-coordinate of lower right corner of the clipping rectangle
- **format** – specifies the method of formatting the text. It can be any combination of the following values (combine using the bitwise OR operator):
  - ◆ DT\_BOTTOM = 8 – Specifies bottom-justified text. This value must be combined with DT\_SINGLELINE.
  - ◆ DT\_CENTER = 1 – Centers text horizontally.
  - ◆ DT\_END\_ELLIPSIS = 32768 or DT\_PATH\_ELLIPSIS = 16384 – Replaces part of the given string with ellipses, if necessary, so that the result fits in the specified rectangle. You can specify DT\_END\_ELLIPSIS to replace characters at the end of the string, or DT\_PATH\_ELLIPSIS to replace characters in the middle of the string. If the string contains backslash (\) characters, DT\_PATH\_ELLIPSIS preserves as much as possible of the text after the last backslash.
  - ◆ DT\_EXPANDTABS = 64 – Expands tab characters. The default number of characters per tab is eight.
  - ◆ DT\_LEFT = 0 – Aligns text flush-left.
  - ◆ DT\_NOCLIP = 256 – Draws without clipping. DrawText is somewhat faster when DT\_NOCLIP is used.
  - ◆ DT\_NOPREFIX = 2048 – Turns off processing of prefix characters. Normally, DrawText interprets the ampersand (&) mnemonic-prefix character as a directive to underscore the character that follows, and the two-ampersand (&&) mnemonic-prefix characters as a directive to print a single ampersand. By specifying DT\_NOPREFIX, this processing is turned off.
  - ◆ DT\_RIGHT = 2 – Aligns text flush-right.
  - ◆ DT\_SINGLELINE = 32 – Specifies single line only. Carriage returns and linefeeds do not break the line.
  - ◆ DT\_TOP = 0 – Specifies top-justified text (single line only).
  - ◆ DT\_VCENTER = 4 – Specifies vertically centered text (single line only).
  - ◆ DT\_WORDBREAK = 16 – Specifies word-breaking. Lines are automatically broken between words if a word would extend past the edge of the rectangle specified by lpRect. A carriage return linefeed sequence will also break the line.

Note: DT\_ constants come from Windows API and are provided here for reference only. They are not defined in AmiBroker therefore you should use numerical values instead.

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**    `// formatted text output sample via low-level gfx functions`

```

CellHeight = 20;
CellWidth = 100;
GfxSelectFont( "Tahoma", CellHeight/2 );

function PrintInCell( string, row, Col )
{
    GfxDrawText( string, Col * CellWidth, row * CellHeight, (Col + 1) *
CellWidth, (row + 1) * CellHeight, 0 );
}

PrintInCell( "Open", 0, 0 );
PrintInCell( "High", 0, 1 );
PrintInCell( "Low", 0, 2 );
PrintInCell( "Close", 0, 3 );
PrintInCell( "Volume", 0, 4 );

GfxSelectPen( colorBlue );
for( i = 1; i < 10 &i < BarCount; i++ )
{
    PrintInCell( StrFormat("%g", O[ i ] ), i, 0 );
    PrintInCell( StrFormat("%g", H[ i ] ), i, 1 );
    PrintInCell( StrFormat("%g", L[ i ] ), i, 2 );
    PrintInCell( StrFormat("%g", C[ i ] ), i, 3 );
    PrintInCell( StrFormat("%g", V[ i ] ), i, 4 );
    GfxMoveTo( 0, i * CellHeight );
    GfxLineTo( 5 * CellWidth, i * CellHeight );
}
GfxMoveTo( 0, i * CellHeight );
GfxLineTo( 5 * CellWidth, i * CellHeight );

for( Col = 1; Col < 6; Col++ )
{
    GfxMoveTo( Col * CellWidth, 0 );
    GfxLineTo( Col * CellWidth, 10 * CellHeight );
}

```

**SEE ALSO**    [GfxSetTextColor\(\)](#) function , [GfxTextOut\(\)](#) function , [GfxSetBkColor\(\)](#) function , [GfxSetBkMode\(\)](#) function

#### References:

The **GfxDrawText** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)

## GFXELLIPSE

### – draw an ellipse

Low-level graphics  
(AFL 3.0)

**SYNTAX**     *GfxEllipse( x1, y1, x2, y2 )*

**RETURNS**    NOTHING

**FUNCTION**    Draws an ellipse. The center of the ellipse is the center of the bounding rectangle specified by x1, y1, x2, and y2. The ellipse is drawn with the current pen, and its interior is filled with the current brush.

Parameters

- x1 – x-coordinate of the upper left corner of the bounding rectangle
- y1 – y-coordinate of the upper left corner of the bounding rectangle
- x2 – x-coordinate of the lower right corner of the bounding rectangle
- y2 – y-coordinate of the lower right corner of the bounding rectangle

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**     *GfxEllipse( 10, 10, 200, 100 );*

**SEE ALSO**     [GfxSelectPen\(\)](#) function , [GfxSelectSolidBrush\(\)](#) function

**References:**

The **GfxEllipse** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)

## **GFXGRADIENTRECT** – draw a rectangle with gradient fill

**Low-level graphics**  
(AFL 3.0)

**SYNTAX**     *GfxGradientRect( x1, y1, x2, y2, fromcolor, tocolor )*

**RETURNS**    NOTHING

**FUNCTION**    Draws a rectangle. The interior of the rectangle is filled using gradient color.

### Parameters

- x1 – x-coordinate of the upper left corner of the rectangle
- y1 – y-coordinate of the upper left corner of the rectangle
- x2 – x-coordinate of the lower right corner of the rectangle
- y2 – y-coordinate of the lower right corner of the rectangle
- fromcolor – the 'upper' color of the gradient fill
- tocolor – the 'lower' color of the gradient fill

The rectangle extends up to, but does not include, the right and bottom coordinates. This means that the height of the rectangle is  $y2 - y1$  and the width of the rectangle is  $x2 - x1$ . Both the width and the height of a rectangle must be greater than 2 and less than 32767.

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**     `GfxGradientRect( 10, 10, 100, 100, colorWhite, colorGrey50 );`

**SEE ALSO**     [GfxRoundRect\(\)](#) function , [GfxRectangle\(\)](#) function

### References:

The **GfxGradientRect** function is used in the following formulas in AFL on-line library:

### More information:

[Updated on-line reference](#)



## **GFXLINETO** – draw a line to specified point

**Low-level graphics**  
(AFL 3.0)

**SYNTAX**     *GfxLineTo( x, y )*

**RETURNS**    NOTHING

**FUNCTION**    Draws a line from the current position up to, but not including, the point specified by x and y. The line is drawn with the selected pen. The current position is set to x,y. Parameters

- x – Specifies the x-coordinate of the end point of the line.
- y – Specifies the y-coordinate of the end point of the line.

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**     `GfxMoveTo( 0, 0 );`  
                  `GfxLineTo( 100, 100 );`

**SEE ALSO**     [GfxMoveTo\(\)](#) function

### **References:**

The **GfxLineTo** function is used in the following formulas in AFL on-line library:

### **More information:**

[Updated on-line reference](#)

## **GFXMOVETO** – move graphic cursor to new position

**Low-level graphics**  
(AFL 3.0)

**SYNTAX**     *GfxMoveTo( x, y )*

**RETURNS**    NOTHING

**FUNCTION**   Moves the current position to the point specified by x and y. Parameters

- x – Specifies the x-coordinate of the new position.
- y – Specifies the y-coordinate of the new position.

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**     *GfxMoveTo( 10, 20 );*

### **SEE ALSO**

### **References:**

The **GfxMoveTo** function is used in the following formulas in AFL on-line library:

### **More information:**

[Updated on-line reference](#)

## GFXPIE

### – draw a pie

**Low-level graphics**  
(AFL 3.0)

**SYNTAX** `GfxPie( x1, y1, x2, y2, x3, y3, x4, y4 )`

**RETURNS** NOTHING

**FUNCTION** Draws a pie-shaped wedge by drawing an elliptical arc whose center and two endpoints are joined by lines. The center of the arc is the center of the bounding rectangle specified by x1, y1, x2, and y2. The starting and ending points of the arc are specified by x3, y3, x4, and y4.

The arc is drawn with the selected pen, moving in a counterclockwise direction. Two additional lines are drawn from each endpoint to the arc's center. The pie-shaped area is filled with the current brush. If x3 equals x4 and y3 equals y4, the result is an ellipse with a single line from the center of the ellipse to the point (x3, y3) or (x4, y4).

#### Parameters

- x1 – x-coordinate of the upper left corner of the bounding rectangle
- y1 – y-coordinate of the upper left corner of the bounding rectangle
- x2 – x-coordinate of the lower right corner of the bounding rectangle
- y2 – y-coordinate of the lower right corner of the bounding rectangle
- x3 – x-coordinate of the arc's starting point. This point does not have to lie exactly on the arc.
- y3 – y-coordinate of the arc's starting point. This point does not have to lie exactly on the arc.
- x4 – x-coordinate of the arc's ending point. This point does not have to lie exactly on the arc.
- y4 – y-coordinate of the arc's ending point. This point does not have to lie exactly on the arc.

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE** `GfxSelectPen( colorRed );`  
`GfxSelectSolidBrush( colorBlue );`  
`GfxPie( 100, 0, 200, 100, 150, 0, 200, 50 );`

**SEE ALSO** [GfxSelectPen\(\)](#) function , [GfxSelectSolidBrush\(\)](#) function

#### References:

The **GfxPie** function is used in the following formulas in AFL on-line library:

#### More information:

[Updated on-line reference](#)

## GFXPOLYGON

### – draw a polygon

**Low-level graphics**  
(AFL 3.0)

**SYNTAX** *GfxPolygon( x1, y1, x2, y2, ... )*

**RETURNS** NOTHING

**FUNCTION** Draws a polygon consisting of two or more points (vertices) connected by lines, using the current pen. The system closes the polygon automatically, if necessary, by drawing a line from the last vertex to the first.

This function takes variable number of arguments and accepts up to 12 points (24 arguments = 12 co-ordinate pairs). The number of arguments must be even as each pair represents (x,y) co-ordinates of the vertex.

The polygon is filled with current brush and outline is painted with current pen.

Parameters:

- x1 – x co-ordinate of first point
- y1 – y co-ordinate of first point
- x2 – x co-ordinate of second point
- y2 – y co-ordinate of second point
- ...
- x12 – x co-ordinate of 12th point
- y12 – y co-ordinate of 12th point

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE** `GfxSelectPen( colorGreen, 2 );`  
`GfxSelectSolidBrush( colorYellow );`  
`GfxPolygon( 250, 200, 200, 200, 250, 0, 200, 50 );`

**SEE ALSO** [GfxPolyline\(\)](#) function , [GfxSelectPen\(\)](#) function , [GfxSelectSolidBrush\(\)](#) function

#### References:

The **GfxPolygon** function is used in the following formulas in AFL on-line library:

#### More information:

[Updated on-line reference](#)

## GFXPOLYLINE

### – draw a polyline

**Low-level graphics**  
(AFL 3.0)

**SYNTAX** `GfxPolyline( x1, y1, x2, y2, ... )`

**RETURNS** NOTHING

**FUNCTION** Draws a set of line segments connecting the points specified by arguments (x1,y1), (x2,y2), ... The lines are drawn from the first point through subsequent points using the current pen. Unlike the GfxLineTo function, the GfxPolyline function neither uses nor updates the current position.

This function takes variable number of arguments and accepts up to 12 points (24 arguments = 12 co-ordinate pairs). The number of arguments must be even as each pair represents (x,y) co-ordinates of the point.

Parameters:

- x1 – x co-ordinate of first point
- y1 – y co-ordinate of first point
- x2 – x co-ordinate of second point
- y2 – y co-ordinate of second point
- ...
- x12 – x co-ordinate of 12th point
- y12 – y co-ordinate of 12th point

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE** `GfxSelectPen( colorGreen, 2 );`  
`GfxPolyline( 250, 200, 200, 200, 250, 0, 200, 50 );`

**SEE ALSO** `GfxPolygon()` function , `GfxSelectPen()` function

**References:**

The **GfxPolyline** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)

**GFXRECTANGLE**  
**– draw a rectangle****Low-level graphics**  
(AFL 3.0)**SYNTAX**     *GfxRectangle( x1, y1, x2, y2 )***RETURNS**     NOTHING**FUNCTION**     Draws a rectangle using the current pen. The interior of the rectangle is filled using the current brush.

Parameters

- x1 – x-coordinate of the upper left corner of the rectangle
- y1 – y-coordinate of the upper left corner of the rectangle
- x2 – x-coordinate of the lower right corner of the rectangle
- y2 – y-coordinate of the lower right corner of the rectangle

The rectangle extends up to, but does not include, the right and bottom coordinates. This means that the height of the rectangle is  $y2 - y1$  and the width of the rectangle is  $x2 - x1$ . Both the width and the height of a rectangle must be greater than 2 and less than 32767.

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**     *GfxRectangle( 10, 10, 30, 30 )***SEE ALSO**     [GfxSelectPen\(\)](#) function , [GfxSelectSolidBrush\(\)](#) function**References:**

The **GfxRectangle** function is used in the following formulas in AFL on-line library:

**More information:**[Updated on-line reference](#)

## GFXROUNDRECT – draw a rectangle with rounded corners

Low-level graphics  
(AFL 3.0)

**SYNTAX**     *GfxRoundRect( x1, y1, x2, y2, x3, y3 )*

**RETURNS**    NOTHING

**FUNCTION**    Draws a rectangle with rounded corners using the current pen. The interior of the rectangle is filled using the current brush.

Parameters

- x1 – x-coordinate of the upper left corner of the rectangle
- y1 – y-coordinate of the upper left corner of the rectangle
- x2 – x-coordinate of the lower right corner of the rectangle
- y2 – y-coordinate of the lower right corner of the rectangle
- x3 – the width of the ellipse used to draw the rounded corners
- y3 – the height of the ellipse used to draw the rounded corners

The rectangle extends up to, but does not include, the right and bottom coordinates. This means that the height of the rectangle is  $y2 - y1$  and the width of the rectangle is  $x2 - x1$ . Both the width and the height of a rectangle must be greater than 2 and less than 32767.

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**     *GfxRoundRect( 10, 10, 100, 100, 15, 15 );*

**SEE ALSO**     [GfxRectangle\(\)](#) function , [GfxSelectPen\(\)](#) function , [GfxSelectSolidBrush\(\)](#) function

**References:**

The **GfxRoundRect** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)

**GFXSELECTFONT****Low-level graphics****– create / select graphic font**

(AFL 3.0)

**SYNTAX**     *GfxSelectFont( "facename", pointsize, weight = fontNormal, italic = False, underline = False, orientation = 0 )*

**RETURNS**    NOTHING

**FUNCTION**    Initializes a font with the specified characteristics. Then selects the the as current for subsequent drawing operations.

Parameters:

- **"facename"** – specifies the typeface name of the font
- **pointsize** – specifies point size of the font (fractional numbers are allowed), for example 11.5 gives 11.5 point font.
- **weight** – specifies the font weight (in inked pixels per 1000). Typical values are: 300 – light, 400 – normal, 700 – bold, 800 – ultrabold
- **italic** – specifies whether the font is italic
- **underline** – specifies whether the font is underlined
- **orientation** – specifies the angle (in 0.1-degree units) between the baseline of a character and the x-axis. The angle is measured counterclockwise from the x-axis.

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**     `GfxSelectFont( "Tahoma", 20, 700 );`  
                   `GfxSetBkMode( 1 );`  
                   `GfxSetTextColor( colorBrown );`  
                   `GfxTextOut( "Testing graphic capabilites", 20, 28 );`

**SEE ALSO**     `GfxLineTo()` function , `GfxMoveTo()` function , `GfxSelectPen()` function ,  
                   `GfxSelectSolidBrush()` function , `GfxSetPixel()` function , `GfxTextOut()` function

**References:**

The **GfxSelectFont** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)



**GFXSELECTPEN****– create / select graphic pen****Low-level graphics**  
(AFL 3.0)**SYNTAX**     *GfxSelectPen( color, width = 1, penstyle = penSolid )***RETURNS**    NOTHING**FUNCTION**    GfxSelectPen initializes (if not already initialized) a pen with the specified style, width, and color. Then selects the pen as current for subsequent drawing operations.

Parameters:

- **color** – specifies color for the pen
- **penstyle** – specifies the style for the pen. Solid=0, Dash=1, Dot=2, Null=5 (invisible pen). Lines of width > 1 can only use solid style. For a list of other possible values, see the Microsoft docs on CreatePen Windows API function.
- **width** – specifies the width of the pen. If this value is 0, the width in device units is always 1 pixel, regardless of the mapping mode (this is useful for drawing hairline lines on printer outputs).

More info on pens in Windows GDI: <http://msdn2.microsoft.com/en-us/library/ms535467>NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**     *GfxSelectPen( colorGreen, 2 );*  
                   *GfxSelectSolidBrush( colorYellow );*  
                   *GfxPolygon( 250, 200, 200, 200, 250, 0, 200, 50 );*

**SEE ALSO**     *GfxLineTo()* function , *GfxMoveTo()* function , *GfxSetPixel()* function , *GfxTextOut()* function**References:**The **GfxSelectPen** function is used in the following formulas in AFL on-line library:**More information:**[Updated on-line reference](#)

## GFXSELECTSOLIDBRUSH

– create / select graphic brush

Low-level graphics  
(AFL 3.0)

**SYNTAX**      *GfxSelectSolidBrush( color )*

**RETURNS**    NOTHING

**FUNCTION**    GfxSelectSolidBrush initializes a brush with a specified solid color. Then selects the brush as current for subsequent drawing operations.

Parameters:

- **color** – specifies color for the brush

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**      `GfxSelectPen( colorGreen, 2 );`  
                  `GfxSelectSolidBrush( colorYellow );`  
                  `GfxPolygon( 250, 200, 200, 200, 250, 0, 200, 50 );`

**SEE ALSO**      [GfxLineTo\(\)](#) function , [GfxMoveTo\(\)](#) function , [GfxSelectPen\(\)](#) function , [GfxSetPixel\(\)](#) function , [GfxTextOut\(\)](#) function

### References:

The **GfxSelectSolidBrush** function is used in the following formulas in AFL on-line library:

### More information:

[Updated on-line reference](#)

## **GFXSETBKCOLOR** – set graphic background color

**Low-level graphics**  
(AFL 3.0)

**SYNTAX**     *GfxSetBkColor( color )*

**RETURNS**    NOTHING

**FUNCTION**    Sets the current background color to the specified color. If the background mode is OPAQUE (see GfxSetBkMode), the system uses the background color to fill the gaps in styled lines, the gaps between hatched lines in brushes, and the background in character cells.

Parameters:

- color – specifies the new background color

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**     GfxSetBkColor( ColorRGB( 10, 20, 30 ) );

**SEE ALSO**     [GfxSelectFont\(\)](#) function , [GfxSetTextColor\(\)](#) function

### **References:**

The **GfxSetBkColor** function is used in the following formulas in AFL on-line library:

### **More information:**

[Updated on-line reference](#)

## **GFXSETBKMODE** – set graphic background mode

**Low-level graphics**  
(AFL 3.0)

**SYNTAX**      *GfxSetBkMode( bkmode )*

**RETURNS**    NOTHING

**FUNCTION**    Sets the background mode. The background mode defines whether the system removes existing background colors on the drawing surface before drawing text, hatched brushes, or any pen style that is not a solid line.

Parameters:

- bkmode – Specifies the mode to be set. This parameter can be either of the following values:  
OPAQUE = 2 – Background is filled with the current background color before the text, hatched brush, or pen is drawn. This is the default background mode.  
TRANSPARENT = 1 – Background is not changed before drawing

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**    `GfxSetBkMode( 1 ); // set transparent mode`

**SEE ALSO**    [GfxSetTextAlign\(\)](#) function , [GfxSetTextColor\(\)](#) function , [GfxTextOut\(\)](#) function , [GfxSelectPen\(\)](#) function

### **References:**

The **GfxSetBkMode** function is used in the following formulas in AFL on-line library:

### **More information:**

[Updated on-line reference](#)

**GFXSETOVERLAYMODE****Low-level graphics**  
(AFL 3.0)**– set low-level graphic overlay mode****SYNTAX**     *GfxSetOverlayMode( mode = 0 )***RETURNS**    NOTHING**FUNCTION**    Sets overlay mode for low-level graphics.

Parameters:

- **mode** – desired overlay mode. Possible values are:
  - 0 – (default) low-level graphic is overlaid on top of charts
  - 1 – charts are overlaid on top of low-level graphics
  - 2 – only low-level graphics is displayed (no charts, no grids, etc)

To learn more about low level graphics please read [Tutorial: Using low-level graphics](#)**EXAMPLE**     *GfxSetOverlayMode( 2 ); // don't display charts nor grids***SEE ALSO**     [GfxArc\(\)](#) function , [GfxChord\(\)](#) function , [GfxCircle\(\)](#) function , [GfxDrawText\(\)](#) function , [GfxEllipse\(\)](#) function , [GfxGradientRect\(\)](#) function , [GfxLineTo\(\)](#) function , [GfxMoveTo\(\)](#) function , [GfxPie\(\)](#) function , [GfxPolygon\(\)](#) function , [GfxPolyline\(\)](#) function , [GfxRectangle\(\)](#) function , [GfxRoundRect\(\)](#) function [\[\[331:Gfx](#)**References:**The **GfxSetOverlayMode** function is used in the following formulas in AFL on-line library:**More information:**[Updated on-line reference](#)

**GFXSETPIXEL****Low-level graphics**  
(AFL 3.0)**– set pixel at specified position to specified color****SYNTAX**     *GfxSetPixel( x, y, color )***RETURNS**     NOTHING**FUNCTION**     Sets the pixel at the specified x, y coordinates to the specified color. The point must be in the visible part drawing surface (otherwise it won't be painted). Parameters

- x – Specifies the x-coordinate of the point.
- y – Specifies the y-coordinate of the point.
- color – specifies the color to be used to paint the point

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**     `GfxSetPixel( 20, 20 );`**SEE ALSO**     [GfxLineTo\(\)](#) function , [GfxMoveTo\(\)](#) function**References:**

The **GfxSetPixel** function is used in the following formulas in AFL on-line library:

**More information:**[Updated on-line reference](#)

## GFXSETTEXTALIGN

### – set text alignment

**Low-level graphics**  
(AFL 3.0)

**SYNTAX** *GfxSetTextAlign( align )*

**RETURNS** NOTHING

**FUNCTION** Sets the text–alignment flags.

The GfxTextOut function uses these flags when positioning a string of text on a display or device. The flags specify the relationship between a specific point and a rectangle that bounds the text. The coordinates of this point are passed as parameters to the TextOut member function. The rectangle that bounds the text is formed by the adjacent character cells in the text string.

Parameters:

- **align** – combination (binary–OR) of one or more the following flags:

X–direction alignment:

- ◆ TA\_CENTER = 6 – Aligns the point with the horizontal center of the bounding rectangle.
- ◆ TA\_LEFT = 0 – Aligns the point with the left side of the bounding rectangle. This is the default setting.
- ◆ TA\_RIGHT = 2 – Aligns the point with the right side of the bounding rectangle.

Y–direction alignment

- ◆ TA\_BASELINE = 24 – Aligns the point with the base line of the chosen font.
- ◆ TA\_BOTTOM = 8 – Aligns the point with the bottom of the bounding rectangle.
- ◆ TA\_TOP = 0 – Aligns the point with the top of the bounding rectangle. This is the default setting.

flags that determine whether the current position is updated when text is written:

- ◆ TA\_NOUPDATECP = 0 – Does not update the current position after each call to a text–output function. This is the default setting.
- ◆ TA\_UPDATECP = 1 – Updates the current x–position after each call to a text–output function. The new position is at the right side of the bounding rectangle for the text. When this flag is set, the coordinates specified in calls to the GfxTextOut member function are ignored

Note: TA\_ constants come from Windows API, they are given for reference only they are not predefined in AmiBroker, so you need to use numerical values.

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE** `GfxSetTextAlign( 6 | 24 ); // center and baseline alignment`

**SEE ALSO** [GfxSetTextColor\(\)](#) function , [GfxTextOut\(\)](#) function , [GfxSetBkColor\(\)](#) function , [GfxSetBkMode\(\)](#) function , [GfxSelectFont\(\)](#) function

**References:**

The **GfxSetTextAlign** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)



**GFXSETTEXTCOLOR**  
**– set graphic text color****Low-level graphics**  
(AFL 3.0)**SYNTAX**     *GfxSetTextColor( color )***RETURNS**    NOTHING**FUNCTION**    Sets the text color to the specified color. AmiBroker will use this text color when writing text to this window using GfxTextOut or GfxDrawText.

The background color for a character is specified by the GfxSetBkColor and GfxSetBkMode member functions.

Parameters:

- color – Specifies the color of the text

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**     GfxSetTextColor( colorRed );  
                  GfxSetTextColor( ColorRGB( 100, 200, 100 ) );

**SEE ALSO**     [GfxTextOut\(\)](#) function , [GfxDrawText\(\)](#) function , [GfxSelectFont\(\)](#) function , [GfxSetBkColor\(\)](#) function , [GfxSetBkMode\(\)](#) function

**References:**

The **GfxSetTextColor** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)

**GFXTEXTOUT****Low-level graphics**  
(AFL 3.0)**– writes text at the specified location****SYNTAX**     *GfxTextOut( "text", x, y )***RETURNS**    NOTHING**FUNCTION**    Writes a character string at the specified location using the currently selected font.

Parameters:

- "text" – Specifies the character string to be drawn
- x – Specifies the x-coordinate of the starting point of the text
- y – Specifies the y-coordinate of the starting point of the text

Character origins are at the upper-left corner of the character cell. By default, the current position is not used or updated by the function.

The font used can be set using GfxSelectFont() function. Text color can be set using GfxSetTextColor() function.

If a formula needs to update the current position when it calls GfxTextOut, the formula can call the GfxSetTextAlign function with flags set to 1 (TA\_UPDATECP Windows flag). When this flag is set, GfxTextOut function ignores the x and y parameters on subsequent calls to GfxTextOut, using the current position instead.

The output of this function is NOT clipped. If you want clip text to user-defined rectangle, use GfxDrawText() function instead.

NOTE: This is LOW-LEVEL graphic function. To learn more about low-level graphic functions please read [TUTORIAL: Using low-level graphics](#).

**EXAMPLE**     `GfxSelectFont( "Times New Roman", 16, 700, True );`  
                   `GfxTextOut( "Percent of shares held by:", 10, 10 );`

**SEE ALSO**     [GfxLineTo\(\)](#) function , [GfxMoveTo\(\)](#) function , [GfxSetPixel\(\)](#) function

**References:**

The **GfxTextOut** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)

**GROUPID****Information / Categories****– get group ID/name**

(AFL 1.8)

**SYNTAX**     *groupid( mode = 0 )***RETURNS**    NUMBER/STRING

**FUNCTION**    Retrieves current stock group ID/name When mode = 0 (the default value ) this function returns numerical group ID (consecutive group number)  
When mode = 1 this function returns name of the group.

**EXAMPLE**     Filter = GroupID() == 7 OR GroupID() == 9;  
AddTextColumn( GroupID( 1 ), "Group name" );

**SEE ALSO****References:**

The **GROUPID** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**HHV****Lowest/Highest****– highest high value****SYNTAX**     *hhv( ARRAY, periods )***RETURNS**     ARRAY**FUNCTION**     Calculates the highest value in the ARRAY over the preceding *periods* (*periods* includes the current day).**EXAMPLE**     The formula "hhv( close, 4)" returns the highest closing price over the preceding four periods; "hhv( high, 8)" returns the highest high price over the preceding eight periods.**SEE ALSO**     [the llv\(\) function \(see Lowest Low Value\).](#)**References:**

The **HHV** function is used in the following formulas in AFL on–line library:

- [% B of Bollinger Bands With Adaptive Zones](#)
- ['D/9E H'](#)
- ['D/9E H'](#)
- [10–20 Indicator](#)
- [30 Week Hi Indicator – Calculate](#)
- [52 Week New High–New Low Index](#)
- [Adaptave Zones O/B &O/S Oscillator](#)
- [Adaptive Price Channel](#)
- [AFL Example](#)
- [AFL Example – Enhanced](#)
- [Against all odds](#)
- [AJDX system](#)
- [An n bar Reversal Indicator](#)
- [Another Fib Level](#)
- [Aroon](#)
- [AR\\_Prediction.afl](#)
- [ATR Study](#)
- [Auto–Optimization Framework](#)
- [Bull Fear / Bear Fear](#)
- [Bullish Percent Index 2 files combined](#)
- [Bullish Percent Index 2004](#)
- [CAMSLIM Cup and Handle Pattern AFL](#)
- [Candle Stick Demo](#)
- [Candlestick Commentary](#)
- [Candlestick Commentary Modified](#)
- [Candlestick Commentary–modified](#)
- [CCI\(20\) Divergence Indicator](#)
- [CCT Kaleidoscope](#)
- [CCT StochasticRSI](#)
- [Chaikin Volume Accumulation](#)
- [Chandelier Exit](#)
- [Chandelier Exit or Advanced Trailing Stop](#)
- [Compare Sectors against Tickers](#)
- [Dahl Oscillator modified](#)

- Dave Landry PullBack Scan
- Dave Landry Pullbacks
- Demand Index
- Divergences
- Donchian Channel
- Double Smoothed Stochastic from W.Bressert
- Double top detection
- DT Oscillator
- Ed Seykota's TSP: Support and Resistance
- Ehlers Fisher Transform
- ekeko price chart
- Elder safe Zone Long + short
- Elder Triple Screen Trading System.
- Elder's SafeZone Stop
- ElderSafeZoneStopLong
- FastStochK FullStochK–D
- Fund Screener
- Gordon Rose
- Head &Shoulders Pattern
- Ichimoku Chart
- Ichimoku charts
- IFT of RSI – Multiple TimeFrames
- Index of 30 Wk Highs Vs Lows
- Kagi Chart
- Larry William's Volatility Channels
- Linear Regression Line &Bands
- MACD and histogram divergence detection
- MACD commentary
- MACD indicator display
- Monthly bar chart
- MultiCycle 1.0
- N–period candlesticks (time compression)
- nikhil
- PFChart – High/Low prices Sept2003
- PFchart with range box sizes
- Pattern – Rectangle Base Breakout on High Vol
- Peterson
- PF Chart – Close – April 2004
- Pivot Finder
- prakash
- Price with Woodies Pivots
- Rainbow Oscillator
- Rea Time Daily Price Levels
- Regression Analysis Line
- Relative Strength Index
- RSI Double–Bottom
- RSI Pointer
- RSIS
- RUTVOL timing signal with BB Scoring routine
- Sainath Sidgiddi
- Steve Woods' Cum. Vol. Float + Cum. Vol. Channels
- Steve Woods' Cumulative Vol. Percentage Indicator

- Stochastic Divergence, negative
- Stochastic Divergences, PDI, NDI
- Stochastic Fast%K and Full
- Stochastic OBV and Price Filter
- Stops Implementation in AFS
- SunI>Support and Resistance
- Support Resistance levels
- TD sequential
- The Stochastic CCI
- Trend Trigger Factor
- Triangle exploration using PFChart
- Triangle search
- Triangular Moving Average new
- Trigonometric Fit – TrigFit with AR for cos / sin
- Vertical Horizontal Filter
- Vertical Horizontal Filter (VHF)
- Volatility System
- Weekly chart
- Weinberg's The Range Indicator
- William's % R
- Williams %R with 9 period signal line
- Williams %R Exploration
- Zig Zag

**More information:**

Updated on–line reference

**HHVBARS****Lowest/Highest****– bars since highest high****SYNTAX**     *hhvbars( ARRAY, periods )***RETURNS**     ARRAY**FUNCTION**     Calculates the number of periods that have passed since the ARRAY reached its *periods* period peak.**EXAMPLE**     The formula "hhvbars( close, 30 )" returns the number of periods that have passed since the closing price reached its 30–period peak.**SEE ALSO****References:**

The **HHVBARS** function is used in the following formulas in AFL on–line library:

- [Aroon Indicators](#)
- [CAMSLIM Cup and Handle Pattern AFL](#)
- [CCI\(20\) Divergence Indicator](#)
- [Cycle Highlighter](#)
- [Cycle Highlighter \(auto best–fit\)](#)
- [Divergences](#)
- [ekeko price chart](#)
- [Fund Screener](#)
- [Gordon Rose](#)
- [MACD and histogram divergence detection](#)
- [Pivot Finder](#)
- [Stochastic Divergence, negative](#)
- [Stochastic Divergences, PDI, NDI](#)
- [Triangle search](#)

**More information:**

[Updated on–line reference](#)

## HIGHEST

– highest value

Lowest/Highest

**SYNTAX**     *highest( ARRAY )*

**RETURNS**     ARRAY

**FUNCTION**     Calculates the highest value in the ARRAY since the first day loaded in the chart.

**EXAMPLE**     The formula `highest( mfi(14) )` returns the highest Money Flow Index value; `highest ( close )` returns the highest closing price.

**SEE ALSO**     [HHV\(\)](#) function , [LOWEST\(\)](#) function , [LLV\(\)](#) function

### References:

The **HIGHEST** function is used in the following formulas in AFL on–line library:

- [Alpha and Beta and R\\_Squared Indicator](#)
- [AR\\_Prediction.afl](#)
- [Auto–Optimization Framework](#)
- [Automatic Trend–line](#)
- [Candle Stick Analysis](#)
- [Candle Stick Demo](#)
- [Double top detection](#)
- [Gordon Rose](#)
- [Market Profile &Market Volume Profile](#)
- [nth \( 1 – 8 \) Order Polynomial Fit](#)
- [Pivot Finder](#)
- [Triangle exploration using PFChart](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

### More information:

[Updated on–line reference](#)



**HIGHESTBARS****Lowest/Highest****– bars since highest value****SYNTAX**     *highestbars( ARRAY )***RETURNS**     ARRAY**FUNCTION**     Calculates the number of periods that have passed since the ARRAY's highest value.**EXAMPLE**     The formula "highestbars( close )" returns the number of periods that have passed since the closing price reached its highest peak.**SEE ALSO****References:**

The **HIGHESTBARS** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**

[Updated on–line reference](#)

**HIGHESTSINCE****Lowest/Highest**  
(AFL 1.4)**– highest value since condition met****SYNTAX**     *highestsince( **EXPRESSION**, **ARRAY**, **Nth = 1** )***RETURNS**     ARRAY**FUNCTION**     Returns the highest ARRAY value since EXPRESSION was true on the Nth most recent occurrence.**EXAMPLE**     highestsince( Cross( macd(), 0 ), Close, 1 ) returns the highest close price since macd() has crossed above zero.**SEE ALSO****References:**

The **HIGHESTSINCE** function is used in the following formulas in AFL on–line library:

- [AJDX system](#)
- [Gann Swing Chart](#)
- [RSI of Weekly Price Array](#)
- [Stochastic of Weekly Price Array](#)
- [Time Frame Weekly Bars](#)
- [Visualization of stoploses and profit in chart](#)
- [Williams Alligator system](#)

**More information:**

[Updated on–line reference](#)

**HIGHESTSINCEBARS****Lowest/Highest**  
(AFL 1.4)**– bars since highest value since condition met****SYNTAX**     *highestsincebars( **EXPRESSION**, **ARRAY**, **Nth = 1** )***RETURNS**     ARRAY**FUNCTION**     Returns the number of bars that have passed since highest ARRAY value since EXPRESSION was true on the Nth most recent occurrence.**EXAMPLE**     highestsincebars( Cross( macd(), 0 ), Close, 1 ) returns the number of bars passed since the highest close price was detected from the time when macd() has crossed above zero.**SEE ALSO****References:**

The **HIGHESTSINCEBARS** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**HOLD**

Trading system toolbox

**– hold the alert signal****SYNTAX**     *hold( **EXPRESSION**, **periods** )***RETURNS**     ARRAY**FUNCTION**     Holds a "true" result of **EXPRESSION** for the specified number of *periods*. This true result is held true over the number of periods specified even if a "false" result is generated.**EXAMPLE**     `hold( cross(rsi(14),70),5 )`**SEE ALSO****References:**

The **HOLD** function is used in the following formulas in AFL on–line library:

- [Peterson](#)
- [TD sequential](#)
- [Williams Alligator system](#)

**More information:**

[Updated on–line reference](#)

**HOURL****Date/Time****– get current bar's hour**

(AFL 2.0)

**SYNTAX**     *hour()***RETURNS**     ARRAY**FUNCTION**     Retrieves current bar's hour**EXAMPLE**     Hour()\*10000 + Minute() \* 100 + Second()**SEE ALSO**     Second(), Minute(), TimeNum()**References:**

The **HOURL** function is used in the following formulas in AFL on–line library:

- [Export Intraday Data](#)
- [Luna Phase](#)

**More information:**

[Updated on–line reference](#)

**IIF****Trading system toolbox****– immediate IF function**

**SYNTAX** `iif( EXPRESSION, TRUE_PART, FALSE_PART )`

**RETURNS** ARRAY

**FUNCTION** "Immediate-IF" – a conditional function that returns the value of the second parameter (TRUE\_PART) if the conditional expression defined by the first parameter (EXPRESSION) is true; otherwise, the value of third parameter is returned (FALSE\_PART). Please note that IIF is a function – so the result of evaluation is returned by that function and should be assigned to some variable. Iif always evaluates both TRUE\_PART and FALSE\_PART, even though it returns only one of them. Because of this, you should watch for undesirable side effects. The following example shows one common error made with IIF function: IIF( condition, result = 7, result = 9 ); // THIS IS WRONG Correct usage is: result = IIF( condition, 7, 9 ); /\* 7 or 9 is \*returned\* and assigned to a variable depending on condition \*/

**EXAMPLE** The formula `result = iif( macd()<signal(), Volume, -Volume)` will assign positive volume values to the result variable on days when macd was below its signal line, and negative volume values on the other days.

**SEE ALSO** More details are given in AFL reference manual (earlier in this chapter)

**Comments:**

<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2003-06-16 03:04:48	IIF can be re-implemented using new if-else flow control statements. The code below shows this and explains what IIF in fact does internally.  <pre>function _IIF( ConditionArray, TrueArray, FalseArray ) { for( i = 0; i &lt; BarCount; i++ ) { if( ConditionArray[ i ] ) { result[ i ] = TrueArray[ i ]; } else { result[ i ] = FalseArray[ i ]; } } }</pre>
<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2003-07-28 09:24:10	If you want to operate on STRINGS use WriteIF function:  <pre>result = WriteIF( condition, "Text 1", "Text 2" );</pre> (note that this function returns single string, depending on 'selected value' of condition).

**References:**

The **IIF** function is used in the following formulas in AFL on-line library:

- % B of Bollinger Bands With Adaptive Zones
- 'D/9E H'
- 'D/9E H'
- 'DE\*H37href='http://www.amibroker.com/library/detail.php?id=413target='\_blank'>3 Price Break
- aboslman2005m
- AC+ acceleration
- accum/dist mov avg crossover SAR
- Adaptave Zones O/B &O/S Oscillator
- ADX Indicator – Colored
- AFL–Excel
- Against all odds
- Alert Output As Quick Rewiev
- Analytic RSI formula
- Andrews Pitchfork
- Andrews PitchforkV3.3
- AO+ Momentum indicator
- Aroon
- AR\_Prediction.afl
- ATR Study
- ATR Trading System
- Auto–Optimization Framework
- Automatic Trend–line
- Balance of Power
- balance of power
- BB squeeze
- Bollinger Band Gap
- Bollinger Fibonacci Bands
- Bull/Bear Volume
- Bullish Percent Index 2 files combined
- Bullish Percent Index 2004
- CAMSLIM Cup and Handle Pattern AFL
- Candle Stick Analysis
- Candle Stick Demo
- Candlestick Commentary
- Candlestick Commentary Modified
- Candlestick Commentary–modified
- Candlestick Volume Bars with Moving Average
- CandleStochastics
- CCI Woodies Style
- CCI(20) Divergence Indicator
- Chande Momentum Oscillator
- Chande's Trend Score
- Chandelier Exit or Advanced Trailing Stop
- Cole
- Color Coded Short Term Reversal Signals
- Color Display.afl
- COMBO
- Compare Sectors against Tickers

- crMathLib
- Cycle Highlighter
- Cycle Highlighter (auto best-fit)
- Dave Landry PullBack Scan
- Dave Landry Pullbacks
- Days to Third Friday
- De Mark's Range Projection
- Demand Index
- DeMarker
- Distance Coefficient Ehlers Filter
- Divergences
- DMI Spread Index
- Dynamic Momentum Index
- Dynamic Momentum Index
- Ed Seykota's TSP: EMA Crossover System
- Ed Seykota's TSP: Support and Resistance
- ekeko price chart
- EKEKO SAR-MF
- Elder Impulse Indicator
- Elder Impulse Indicator V2
- Elder safe Zone Long + short
- Elder's Market Thermometer
- Elder's SafeZone Stop
- ElderSafeZoneStopLong
- ElderSafeZoneStopShort
- End Of Year Trading
- Expiry Thursday for Indian markets
- FastStochK FullStochK-D
- Fib CMO
- Follow the Leader
- Fund Screener
- Gann HiLo Indicator and System
- Gann Swing Chart
- garythompson
- garythompson
- Gordon Rose
- half-automated Trading System
- Head & Shoulders Pattern
- Hilbert Study
- Hull Moving Average
- Hurst "Like" DE
- Hurst Constant
- IFT of RSI – Multiple TimeFrames
- Indicator Explorer (ZigZag)
- INTRADAY HEIKIN ASHI new
- IntraDay Open Marker
- Kagi Chart
- Lagging MA-Xover
- Larry William's Volatility Channels
- lastNDaysBeforeDate
- Linear Regression Line w/ Std Deviation Channels
- Luna Phase



- LunarPhase
- MA Difference 20 Period
- MACD and histogram divergence detection
- MACD indicator display
- Main price chart with Rainbow & SAR
- Market Profile & Market Volume Profile
- Modified Darvas Box
- Monte Carlo
- Monthly bar chart
- Moving Average "Crash" Test
- Moving Averages NoX
- MultiCycle 1.0
- N-period candlesticks (time compression)
- nikhil
- Nonlinear Ehlers Filter
- Noor\_Doodie
- nth ( 1 – 8 ) Order Polynomial Fit
- Option Calls, Puts and days till third friday.
- PFChart – High/Low prices Sept2003
- PFchart with range box sizes
- Performance Check
- Performance Overview
- Peterson
- PF Chart – Close – April 2004
- Pivot Finder
- Pivots for Intraday Forex Charts
- Polarized Fractal Efficiency
- Position Sizing and Risk Price Graph
- Position Sizing and Risk Price Graph – 2
- Price with Woodies Pivots
- Price–Volume Rank
- QP2 Float Analysis
- R–Squared
- Rainbow Oscillator
- Range Expansion Index
- Ranking Ticker WatchList
- Raw ADX
- Rea Time Daily Price Levels
- Regression Analysis Line
- Relative Strength Multichart of up to 10 tickers
- Relative Vigour Index
- ROC of MACD Weekly
- RSI of Weekly Price Array
- RSI Trendlines and Wedges
- RSIS
- RUTVOL timing signal with BB Scoring routine
- Sainath Sidgiddi
- Schiff Lines
- SectorRSI
- Standard Error Bands (Native AFL)
- STD\_STK Multi
- Steve Woods' Cum. Vol. Float + Cum. Vol. Channels

- Steve Woods' Cumulative Vol. Percentage Indicator
- Stochastic Divergence, negative
- Stochastic Divergence, positive
- Stochastic Divergences, PDI, NDI
- Stochastic Fast%K and Full
- Stochastics Trendlines
- SunI>SUPER PIVOT POINTS
- Support and Resistance
- TD sequential
- The Fibonacci behavior
- The Mean RSI
- The Mean RSI (variations)
- The Saturation Indicator D\_sat
- Time Left in Bar
- Time segment value
- tomy\_frenchy
- Trading ATR 10–1
- Trend Continuation Factor
- Trend Detection
- Trend exploration with multiple timeframes
- Trending Ribbon
- Triangle exploration using PFChart
- Triangular Moving Average
- Triangular Moving Average new
- Trigonometric Fit – TrigFit with AR for cos / sin
- Twiggs Money Flow
- TWS auto–export Executions–file parser
- Using From and To dates from Auto Analysis in Code
- Vic Hiebner
- Visualization of stoploses and profit in chart
- Volatility System
- Volume – compared with Moving Avg (100%)
- Volume Oscillator
- Weekly chart
- Weekly Trend in Daily Graph
- Weinberg's The Range Indicator
- Williams Alligator system
- Woodies CCI
- Zig Zag
- Zig Zag Indicator with Valid Entry and Exit Points
- \_Volume

**More information:**

Updated on–line reference

**INDUSTRYID****Information / Categories****– get industry ID / name**

(AFL 1.8)

**SYNTAX**      *industryid( mode = 0 )***RETURNS**     NUMBER/STRING

**FUNCTION**     Retrieves current stock industry ID/name When mode = 0 (the default value ) this function returns numerical industry ID (consecutive industry number)  
When mode = 1 this function returns name of the industry.

**EXAMPLE**      Filter = IndustryID() == 7 OR IndustryID() == 9;  
AddTextColumn( IndustryID( 1 ), "Industry name" );

**SEE ALSO****References:**

The **INDUSTRYID** function is used in the following formulas in AFL on–line library:

- [Dave Landry PullBack Scan](#)
- [Elder Triple Screen Trading System.](#)
- [MACD and histogram divergence detection](#)

**More information:**

[Updated on–line reference](#)

**INSIDE****Basic price pattern detection****– inside day****SYNTAX**     *inside()***RETURNS**     ARRAY

**FUNCTION**     Gives a "1" or "true" when an inside day occurs. Gives "0" otherwise. An inside day occurs when today's high is less than yesterday's high and today's low is greater than yesterday's low.

**EXAMPLE****SEE ALSO****References:**

The **INSIDE** function is used in the following formulas in AFL on–line library:

- [Gann Five Day pullback](#)
- [Gann Swing Chart](#)
- [Vic Huebner](#)

**More information:**

[Updated on–line reference](#)

**INT****Math functions****– integer part**

**SYNTAX**     *int( NUMBER )*  
                  *int( ARRAY )*

**RETURNS**    NUMBER  
                  ARRAY

**FUNCTION**   Removes the fractional portion of NUMBER or ARRAY and returns the integer part.

**EXAMPLE**    The formula "int( 10.7 )" returns 10; the formula "int(–19.8 )" returns –19.

**SEE ALSO**    The ceil() function; the floor() function; the frac() function.

**References:**

The **INT** function is used in the following formulas in AFL on–line library:

- ['D/9E H'](#)
- ['D/9E H'](#)
- [AFL Timing functions](#)
- [Baseline Relative Performance Watchlist charts V2](#)
- [Bullish Percent Index 2 files combined](#)
- [Candle Stick Demo](#)
- [Color Display.afl](#)
- [Cycle Highlighter](#)
- [Cycle Highlighter \(auto best–fit\)](#)
- [Gartley 222 Pattern Indicator](#)
- [Hull Moving Average](#)
- [Hurst "Like" DE](#)
- [Hurst Constant](#)
- [Kagi Chart](#)
- [Log Time Scale](#)
- [LunarPhase](#)
- [Modified Momentum Finder DDT–NB](#)
- [Randomize\(\)](#)
- [Standard Error Bands](#)
- [Triangle exploration using PFChart](#)
- [VAMA](#)

**More information:**

[Updated on–line reference](#)

**INTERVAL****Date/Time**  
(AFL 2.1)**– get bar interval (in seconds)****SYNTAX**     *interval( format = 0 )***RETURNS**     NUMBER**FUNCTION**     Interval() function returns bar interval.

Possible formats: format = 0 – returns bar interval in seconds  
 format = 1 – as above plus TICK bar intervals are returned with negative sign so Interval() function applied to 10 tick chart will return –10  
 format = 2 – returns STRING with name of interval such as "weekly/monthly/daily/hourly/15-minute/5-tick"

Example time intervals in seconds:

tick bars = 0

5 sec bars = 5

1 min bars = 60 (inMinute constant)

hourly bars = 3600

daily bars = 86400 (inDaily constant)

weekly bars = 432001 (inWeekly constant)

monthly bars = 2160001 (inMonthly constant)

**EXAMPLE**     "Interval in seconds " + WriteVal( Interval() );**SEE ALSO****References:**

The **INTERVAL** function is used in the following formulas in AFL on–line library:

- ['D/9E H'](#)
- ['D/9E H'](#)
- [AFL Timing functions](#)
- [CCI Woodies Style](#)
- [DPO with shading](#)
- [ekeko price chart](#)
- [MACD and histogram divergence detection](#)
- [Price with Woodies Pivots](#)
- [Rea Time Daily Price Levels](#)
- [Time Left in Bar](#)

**More information:**

[Updated on–line reference](#)

**INWATCHLIST**

Information / Categories

**– watch list membership test (by ordinal number)****SYNTAX**     *InWatchList( listno )***RETURNS**     NUMBER**FUNCTION**     Checks if the stock belongs to a watch list number *listno*. If yes – the function returns 1 otherwise 0.**EXAMPLE**     Filter= InWatchList( 3 ) OR InWatchList( 5 );**SEE ALSO**     [InWatchListName\(\)](#) function**References:**

The **InWatchList** function is used in the following formulas in AFL on–line library:

- [Count Tickers in Watchlist](#)

**More information:**

[Updated on–line reference](#)

**INWATCHLISTNAME****Information / Categories****– watch list membership test (by name)**

(AFL 3.0)

**SYNTAX**     *InWatchList( "name" )***RETURNS**    NUMBER**FUNCTION**    Checks if the stock belongs to a watch list number "*listname*". If yes – the function returns 1 otherwise 0.**EXAMPLE**     Filter= InWatchListName( "My Hotlist" ) OR InWatchList( "My Second Hotlist" );**SEE ALSO**     [InWatchList\(\)](#) function**References:**

The **InWatchListName** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)



**ISCONTINUOUS****Information / Categories****– checks 'continuous quotations' flag state**

(AFL 2.60)

**SYNTAX**     *IsContinuous()***RETURNS**    NUMBER**FUNCTION**    Returns 1 if current symbol has 'continuous quotations' flag turned on in Symbol->Information window. Returns zero otherwise.**EXAMPLE****SEE ALSO****References:**

The **IsContinuous** function is used in the following formulas in AFL on-line library:

**More information:**[Updated on-line reference](#)

**ISEMPTY****– empty value check****Miscellaneous functions**  
(AFL 1.5)**SYNTAX**     *isempty( ARRAY )***RETURNS**     ARRAY

**FUNCTION**     returns 1 (or 'true') when given point in array is {empty}  
 Note: {empty} value is used internally by AFL to mark bars when the value is not available – for example for the first 20 bars the value of 20–day simple moving average is not available ({empty})

IsNull is a synonym for IsEmpty. It is suggested to use IsNull in new formulas, because of naming consistency with Null constant.

**EXAMPLE**     `movavgv = ma( close, 30 );`  
                  `WriteIF( IsEmpty( movavgv ), "Moving average not available yet",`  
                  `WriteVal( movavgv ) );`

**SEE ALSO**     [ISEMPTY\(\)](#) function , [ISNAN\(\)](#) function , [ISNULL\(\)](#) function , [ISTRUE\(\)](#) function

**References:**

The **ISEMPTY** function is used in the following formulas in AFL on–line library:

- [AFL Timing functions](#)
- [Elder safe Zone Long + short](#)
- [Fund Screener](#)
- [lastNDaysBeforeDate](#)
- [MACD indicator display](#)
- [Performance Overview](#)
- [QP2 Float Analysis](#)
- [Weekly chart](#)
- [Zig Zag](#)

**More information:**

[Updated on–line reference](#)

**ISFAVORITE****Information / Categories****– check if current symbol belongs to favorites**

(AFL 2.5)

**SYNTAX**     *IsFavorite()***RETURNS**    NUMBER**FUNCTION**    The IsFavorite function returns True (1) if current symbol belongs to favorites, returns False (0) otherwise.

**EXAMPLE**     `if( IsFavorite() )`  
                  `{`  
                  `printf( Name() + " belongs to favourites " );`  
                  `}`

**SEE ALSO**     `IsIndex()` function**References:**

The **IsFavorite** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**ISFINITE****Miscellaneous functions**  
(AFL 2.3)**– check if value is not infinite****SYNTAX**     *IsInfinite( x )***RETURNS**    NUMBER, ARRAY**FUNCTION**    returns a nonzero value (1 or TRUE) if its argument x is not infinite, that is, if  $INF < x < +INF$ . It returns 0 (FALSE) if the argument is infinite or a NaN.

x can be number or array

**EXAMPLE**     IsFinite( 1/0 );**SEE ALSO**     [NZ\(\)](#) function , [ISNAN\(\)](#) function**References:**The **ISFINITE** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [tomy\\_frenchy](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**[Updated on–line reference](#)

**ISINDEX****Information / Categories****– check if current symbol is an index**

(AFL 2.5)

**SYNTAX**     *IsIndex()***RETURNS**    NUMBER**FUNCTION**    The IsIndex function returns True (1) if current symbol is an index, returns False (0) otherwise.

**EXAMPLE**     `if( IsIndex() )`  
                  `{`  
                  `printf( Name() + " is an index" );`  
                  `}`

**SEE ALSO**     [IsFavorite\(\)](#) function**References:**

The **IsIndex** function is used in the following formulas in AFL on–line library:

- [MACD and histogram divergence detection](#)

**More information:**

[Updated on–line reference](#)

**ISNAN****– checks for NaN (not a number)****Miscellaneous functions**  
(AFL 2.3)**SYNTAX**     *IsNan( x )***RETURNS**    NUMBER, ARRAY**FUNCTION**    Returns a nonzero value (1 or TRUE) if the argument x is a NaN; otherwise it returns 0 (FALSE). A NaN is generated when the result of a floating-point operation cannot be represented in Institute of Electrical and Electronics Engineers (IEEE) format.**EXAMPLE**     IsNan( 0/0 );**SEE ALSO**     [NZ\(\)](#) function**References:**

The **ISNAN** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [tomy\\_frenchy](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**[Updated on–line reference](#)

## ISNULL – check for Null (empty) value

Miscellaneous functions  
(AFL 2.3)

**SYNTAX**     *IsNull( x )*

**RETURNS**    NUMBER, ARRAY

**FUNCTION**    this function is synonym of IsEmpty(). Gives True if value is equal to Null (empty) value.

**EXAMPLE**     `movavg = ma( close, 30 );  
WriteIF( IsNull( movavg ), "Moving average not available yet",  
WriteVal( movavg ) );`

**SEE ALSO**     [ISEMPTY\(\)](#) function

### References:

The **ISNULL** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [Auto–Optimization Framework](#)
- [Chandelier Exit](#)
- [tomy\\_frenchy](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

### More information:

[Updated on–line reference](#)

**ISTRUE****– true value (non-empty and non-zero) check****Miscellaneous functions**  
(AFL 1.5)**SYNTAX**     *isttrue( ARRAY )***RETURNS**     ARRAY**FUNCTION**     returns 1 (or 'true') when given point is not {empty} AND not zero**EXAMPLE****SEE ALSO****References:**

The **ISTRUE** function is used in the following formulas in AFL on-line library:

- [Cycle Highlighter \(auto best-fit\)](#)
- [FirstBarIndex\(\), LastBarIndex\(\)](#)
- [MACD and histogram divergence detection](#)
- [Relative Strength Multichart of up to 10 tickers](#)

**More information:**

[Updated on-line reference](#)



**LASTVALUE**

Trading system toolbox

**– last value of the array****SYNTAX**     *lastvalue*(**ARRAY**, *lastmode* = **True** )**RETURNS**    NUMBER**FUNCTION**   Returns last calculated value of the specified **ARRAY**. The result of this function can be used in place of a constant (**NUMBER**) in any function argument.If **ARRAY** is undefined (e.g., only 100–days loaded and you request the last value of a 200–day moving average) then the *lastvalue* function returns zero.**Caveat:** since this function fills an entire data array with the last value of another array, it allows a formula to look into the future.*lastmode* parameter:**(affects only commentary/interpretation)**When it is **True** – then true last value is used alwaysif it is **False** – then in commentary the 'selected' value is returned

In pre–4.08.1 versions commentary/interpretation/tooltip code evaluation was somewhat special because *LastValue* returned in fact not the last but the value of array at selected point (by vertical line or by tooltip) This caused some problems in displaying indicator values that used *LastValue* in its construction. To address this now *LastValue* used in Commentaries by default returns true last value. So you should modify your existing commentary/interpretation code that used *LastValue* to use now *SelectedValue*( array ) function to maintain the same behaviour. Alternatively you can use *LastValue*( array, 0 ).

**EXAMPLE****SEE ALSO**    [SELECTEDVALUE\(\)](#) function**References:**The **LASTVALUE** function is used in the following formulas in AFL on–line library:

- 'D/9E H'
- 'D/9E H'
- 3 Price Break
- Alpha and Beta and R\_Squared Indicator
- AR\_Prediction.afl
- Auto–Optimization Framework
- Automatic Trend–line
- Bullish Percent Index 2 files combined
- Candle Stick Analysis
- Candle Stick Demo
- CCI(20) Divergence Indicator
- Color Display.afl
- Cycle Highlighter
- Cycle Highlighter (auto best–fit)
- Dave Landry PullBack Scan
- DMI Spread Index
- ekeko price chart
- Elder Triple Screen Trading System.

- FirstBarIndex(), LastBarIndex()
- Futures – Dollar Move Indicator
- Gann Swing Chart
- Gordon Rose
- Head & Shoulders Pattern
- Hurst "Like" DE
- Hurst Constant
- Kagi Chart
- Linear Regression Line & Bands
- Linear Regression Line w/ Std Deviation Channels
- MACD and histogram divergence detection
- MACD commentary
- MACD indicator display
- Modified Momentum Finder DDT–NB
- Monthly bar chart
- Moving Average "Crash" Test
- Moving Averages NoX
- Multiple sinus noised
- N–period candlesticks (time compression)
- nth ( 1 – 8 ) Order Polynomial Fit
- PFChart – High/Low prices Sept2003
- Pattern Recognition Exploration
- PF Chart – Close – April 2004
- Pivot Finder
- Pivot Point with S/R Trendlines
- Price Persistency
- QP2 Float Analysis
- Ranking and sorting stocks
- Real Time Daily Price Levels
- Regression Analysis Line
- Relative Strength Multichart of up to 10 tickers
- Renko Chart
- RSI Trendlines and Wedges
- STD\_STK Multi
- Steve Woods' Cum. Vol. Float + Cum. Vol. Channels
- Steve Woods' Cumulative Vol. Percentage Indicator
- Stochastics Trendlines
- StochD\_StochK Single.afl
- SUPER PIVOT POINTS
- Support Resistance levels
- The Fibonacci behavior
- Time Left in Bar
- Tom DeMark Trend Lines
- Triangle exploration using PFChart
- Triangle search
- Trigonometric Fit – TrigFit with AR for cos / sin
- Using From and To dates from Auto Analysis in Code
- Weekly chart
- Williams Alligator system
- WLBuidProcess
- Zig Zag Indicator with Valid Entry and Exit Points

**More information:**

[Updated on-line reference](#)

**LINEARRAY****Exploration / Indicators****– generate trend–line array**

(AFL 2.5)

**SYNTAX**     *LineArray( x0, y0, x1, y1, extend = 0, usebarindex = False )***RETURNS**     ARRAY**FUNCTION**     The **LineArray** function generates array equivalent to trend line drawn from point (x0, y0) to point (x1, y1). x coordinates are in bars (zero based), y coordinates are in dollars.

Note: x0 must be SMALLER than x1.

Note 2: the function accepts only numbers therefore generates single line. To produce multiple lines you have to call it many times with different co–ordinates.

*extend* parameter controls automatic extension of the trend line: if extend is 1 then line is right extended. if extend is 2 then line is left extended if extend is 3 then line is left and right extended*usebarindex* parameter controls if x coordinates are interpreted as current array indexes (from 0..BarCount–1) (when *usebarindex* = False) or as absolute bar indexes (returned by BarIndex() function) when *usebarindex* = True. These two may differ if QuickAFL feature is turned on.

**EXAMPLE**

```

y0=LastValue( Trough( L, 5, 2 ) );
y1=LastValue( Trough( L, 5, 1 ) );
x0=BarCount - 1 - LastValue( TroughBars( L, 5, 2 ) );
x1=BarCount - 1 - LastValue( TroughBars( L, 5, 1 ) );
Line = LineArray( x0, y0, x1, y1, 1 );
Plot( C, "C", colorWhite, styleCandle );
Plot( Line, "Trend line", colorBlue );

```

**SEE ALSO****References:**The **LineArray** function is used in the following formulas in AFL on–line library:

- [3 Price Break](#)
- [Another Fib Level](#)
- [CCI\(20\) Divergence Indicator](#)
- [Gartley 222 Pattern Indicator](#)
- [Gordon Rose](#)
- [PF Chart – Close – April 2004](#)
- [Pivot Point with S/R Trendlines](#)
- [Rea Time Daily Price Levels](#)
- [Tom DeMark Trend Lines](#)

**More information:**[Updated on–line reference](#)

**LINEARREG****Statistical functions****– linear regression end–point**

(AFL 2.2)

**SYNTAX**     *LinearReg( ARRAY, periods )***RETURNS**     ARRAY**FUNCTION**     Calculates linear regression line end–point value according to  $a + b * x$  (where a and b are intercept and slope of linear regression line) from the ARRAY using *periods* range.**EXAMPLE**     `LinearReg( close, 10 );`**SEE ALSO****References:**

The **LINEARREG** function is used in the following formulas in AFL on–line library:

- [CCI Woodies Style](#)
- [Moving Trend Bands \(MTB\)](#)
- [Price with Woodies Pivots](#)
- [Standard Error Bands \(Native AFL\)](#)
- [Trend Detection](#)

**More information:**[Updated on–line reference](#)

**LINREGINTERCEPT****Statistical functions**  
(AFL 2.2)

–

**SYNTAX**     *LinRegIntercept( ARRAY, periods )***RETURNS**     ARRAY**FUNCTION**     Calculates intercept of linear regression line – the "a" coefficient in  $a + b \cdot x$  (LinRegSlope calculates b) from the ARRAY using *periods* range.

**EXAMPLE**     `x = Cum(1);`  
                   `lastx = LastValue( x ); Daysback = 10; aa = LastValue( LinRegIntercept( Close, Daysback ) );`  
                   `bb = LastValue( LinRegSlope( Close, Daysback ) );`

`y = Aa + bb * ( x – (Lastx – DaysBack) ); Plot( Close, "Close", colorBlack, styleCandle );`  
                   `Plot( If( x >= (lastx – Daysback), y, –1e10 ), "LinReg", colorRed );`

**SEE ALSO****References:**

The **LINREGINTERCEPT** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [Dave Landry PullBack Scan](#)
- [Linear Regression Line w/ Std Deviation Channels](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**

[Updated on–line reference](#)

**LINREGSLOPE****Statistical functions****– linear regression slope**

(AFL 1.4)

**SYNTAX**     *linregslope( ARRAY, periods )***RETURNS**     ARRAY**FUNCTION**     Calculates linear regression line slope from the ARRAY using *periods* range.

**EXAMPLE**     `x = Cum(1);`  
                   `lastx = LastValue( x ); Daysback = 10; aa = LastValue( LinRegIntercept( Close, Daysback ) );`  
                   `bb = LastValue( LinRegSlope( Close, Daysback ) );`

`y = Aa + bb * ( x - (Lastx - DaysBack) ); Plot( Close, "Close", colorBlack, styleCandle );`  
                   `Plot( If( x >= (lastx - Daysback), y, -1e10 ), "LinReg", colorRed );`

**SEE ALSO****References:**

The **LINREGSLOPE** function is used in the following formulas in AFL on-line library:

- ['DE\\*H37href='http://www.amibroker.com/library/detail.php?id=718PE'](http://www.amibroker.com/library/detail.php?id=718PE)  
   [target='\\_blank'>abosliman2005m](#)
- [AR\\_Prediction.afl](#)
- [ATR Study](#)
- [CCT Kaleidoscope](#)
- [Dave Landry PullBack Scan](#)
- [ekeko price chart](#)
- [Gann Five Day pullback](#)
- [Linear Regression Line w/ Std Deviation Channels](#)
- [prakash](#)
- [R-Squared](#)
- [Regression Analysis Line](#)
- [RSIS](#)
- [Trend Analysis\\_Comentary](#)
- [Trend exploration with multiple timeframes](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [Visualization of stoploses and profit in chart](#)

**More information:**

[Updated on-line reference](#)

**LLV****Lowest/Highest****– lowest low value****SYNTAX**     *llv( ARRAY, periods )***RETURNS**     ARRAY**FUNCTION**     Calculates the lowest value in the ARRAY over the preceding *periods* (*periods* includes the current day).**EXAMPLE**     The formula "llv( close, 14 )" returns the lowest closing price over the preceding 14 periods.**SEE ALSO**     The hhv() function (see Highest High Value ).**References:**

The **LLV** function is used in the following formulas in AFL on–line library:

- [% B of Bollinger Bands With Adaptive Zones](#)
- ['D/9E H'](#)
- ['D/9E H'](#)
- [10–20 Indicator](#)
- [30 Week Hi Indicator – Calculate](#)
- [52 Week New High–New Low Index](#)
- [Adaptave Zones O/B &O/S Oscillator](#)
- [Adaptive Price Channel](#)
- [Against all odds](#)
- [AJDX system](#)
- [An n bar Reversal Indicator](#)
- [Another Fib Level](#)
- [Aroon](#)
- [AR\\_Prediction.afl](#)
- [ATR Study](#)
- [Auto–Optimization Framework](#)
- [Bull Fear / Bear Fear](#)
- [Bullish Percent Index 2 files combined](#)
- [Bullish Percent Index 2004](#)
- [CAMSLIM Cup and Handle Pattern AFL](#)
- [Candle Stick Demo](#)
- [Candlestick Commentary](#)
- [Candlestick Commentary Modified](#)
- [Candlestick Commentary–modified](#)
- [CCI\(20\) Divergence Indicator](#)
- [CCT Kaleidoscope](#)
- [CCT StochasticRSI](#)
- [Chaikin Volume Accumulation](#)
- [Chandelier Exit](#)
- [Chandelier Exit or Advanced Trailing Stop](#)
- [Compare Sectors against Tickers](#)
- [Dahl Oscillator modified](#)
- [Dave Landry PullBack Scan](#)
- [Dave Landry Pullbacks](#)
- [Demand Index](#)



- Divergences
- Donchian Channel
- Double Smoothed Stochastic from W.Bressert
- DT Oscillator
- Ed Seykota's TSP: Support and Resistance
- Ehlers Fisher Transform
- Elder safe Zone Long + short
- Elder Triple Screen Trading System.
- ElderSafeZoneStopShort
- FastStochK FullStochK-D
- Fund Screener
- Gordon Rose
- Head &Shoulders Pattern
- Ichimoku Chart
- Ichimoku charts
- IFT of RSI – Multiple TimeFrames
- Index of 30 Wk Highs Vs Lows
- Kagi Chart
- Larry William's Volatility Channels
- MACD and histogram divergence detection
- MACD commentary
- MACD indicator display
- Monthly bar chart
- MultiCycle 1.0
- N-period candlesticks (time compression)
- nikhil
- NR4 Historical Volatility System
- PFChart – High/Low prices Sept2003
- PFchart with range box sizes
- Pattern – Rectangle Base Breakout on High Vol
- Peterson
- PF Chart – Close – April 2004
- Pivot Finder
- prakash
- Price with Woodies Pivots
- Pullback System No. 1
- Rainbow Oscillator
- Rea Time Daily Price Levels
- Regression Analysis Line
- Relative Strength Index
- RSI Double-Bottom
- RSI Pointer
- RSIS
- RUTVOL timing signal with BB Scoring routine
- Sainath Sidgiddi
- Steve Woods' Cum. Vol. Float + Cum. Vol. Channels
- Stochastic Divergence, positive
- Stochastic Divergences, PDI, NDI
- Stochastic Fast%K and Full
- Stochastic OBV and Price Filter
- Stops Implementation in AFS
- Sunl>Support and Resistance

- Support Resistance levels
- TD sequential
- The Stochastic CCI
- Trend Trigger Factor
- Triangle exploration using PFChart
- Triangle search
- Trigonometric Fit – TrigFit with AR for cos / sin
- Vertical Horizontal Filter
- Vertical Horizontal Filter (VHF)
- Volatility System
- Weekly chart
- Weinberg's The Range Indicator
- William's % R
- Williams %R Exploration
- Zig Zag

**More information:**

[Updated on-line reference](#)

**LLVBARS****Lowest/Highest****– bars since lowest low****SYNTAX**     *llvbars( ARRAY, periods )***RETURNS**     ARRAY**FUNCTION**     Calculates the number of periods that have passed since the ARRAY reached its *periods* period trough.**EXAMPLE**     The formula "llvbars( close,50 )" returns the number of periods that have passed since the closing price reached its 50 period trough.**SEE ALSO****References:**

The **LLVBARS** function is used in the following formulas in AFL on–line library:

- [Aroon Indicators](#)
- [CAMSLIM Cup and Handle Pattern AFL](#)
- [CCI\(20\) Divergence Indicator](#)
- [Cycle Highlighter](#)
- [Cycle Highlighter \(auto best–fit\)](#)
- [Divergences](#)
- [ekeko price chart](#)
- [Fund Screener](#)
- [Gordon Rose](#)
- [Linear Regression Line & Bands](#)
- [MACD and histogram divergence detection](#)
- [Pivot Finder](#)
- [Stochastic Divergence, positive](#)
- [Stochastic Divergences, PDI, NDI](#)
- [Triangle search](#)

**More information:**

[Updated on–line reference](#)

**LOG****Math functions****– natural logarithm**

**SYNTAX**      *log( NUMBER )*  
                  *log( ARRAY )*

**RETURNS**    NUMBER  
                  ARRAY

**FUNCTION**    Calculates the natural logarithm of NUMBER or ARRAY.

**EXAMPLE**

**SEE ALSO**    exp() Exponential function

**Comments:**

<b>Tomasz Janeczko</b>	The synonym to 'log' is 'ln' function.
2006-03-02 04:26:40	

**References:**

The **LOG** function is used in the following formulas in AFL on-line library:

- ['D/9E H'](#)
- ['D/9E H'](#)
- [Andrews Pitchfork](#)
- [Andrews PitchforkV3.3](#)
- [AR\\_Prediction.afl](#)
- [CAMSLIM Cup and Handle Pattern AFL](#)
- [Dave Landry PullBack Scan](#)
- [Ehler's filters and indicators](#)
- [Ehlers Fisher Transform](#)
- [Elder Triple Screen Trading System.](#)
- [Historical Volatility Scan – 6/100](#)
- [Historical Volatility Scan – 50 Day](#)
- [Hurst Constant](#)
- [IFT of RSI – Multiple TimeFrames](#)
- [MultiCycle 1.0](#)
- [NR4 Historical Volatility System](#)
- [Probability Calculator](#)
- [Schiff Lines](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [Volume Oscillator](#)
- [Zig Zag](#)

**More information:**

[Updated on-line reference](#)

**LOG10****Math functions****– decimal logarithm**

**SYNTAX**     *log10( NUMBER )*  
                 *log10( ARRAY )*

**RETURNS**    NUMBER  
                 ARRAY

**FUNCTION**    Calculates the decimal logarithm of NUMBER or ARRAY.

**EXAMPLE**

**SEE ALSO**     .

**References:**

The **LOG10** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**

[Updated on–line reference](#)

**LOWEST**  
– lowest value**Lowest/Highest****SYNTAX**     *lowest*( **ARRAY** )**RETURNS**     ARRAY**FUNCTION**     Calculates the lowest value in the ARRAY.**EXAMPLE**     The formula `lowest( rsi(14) )`; returns the lowest Relative Strength Index value ;  
                 `lowest ( close )` returns the lowest closing price.**SEE ALSO**     [HHV\(\)](#) function , [LLV\(\)](#) function , [HIGHEST\(\)](#) function**References:**

The **LOWEST** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [Candle Stick Analysis](#)
- [Candle Stick Demo](#)
- [Multiple sinus noised](#)
- [Triangle exploration using PFChart](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**[Updated on–line reference](#)

**LOWESTBARS****Lowest/Highest****– bars since lowest****SYNTAX**     *lowestbars( ARRAY )***RETURNS**     ARRAY**FUNCTION**     Calculates the number of periods that have passed since the ARRAY's lowest value.**EXAMPLE**     The formula "lowestbars( close )" returns the number of periods that have passed since the closing price reached its lowest point.**SEE ALSO****References:**

The **LOWESTBARS** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**LOWESTSINCE****Lowest/Highest**  
(AFL 1.4)**– lowest value since condition met****SYNTAX**     *lowestsince( **EXPRESSION**, **ARRAY**, **Nth = 1** )***RETURNS**     ARRAY**FUNCTION**     Returns the lowest ARRAY value since EXPRESSION was true on the Nth most recent occurrence.**EXAMPLE**     lowestsince( Cross( macd(), 0 ), Close, 1 ) returns the lowest close price since macd() has crossed above zero.**SEE ALSO****References:**

The **LOWESTSINCE** function is used in the following formulas in AFL on–line library:

- [Gann Swing Chart](#)
- [RSI of Weekly Price Array](#)
- [Stochastic of Weekly Price Array](#)
- [Time Frame Weekly Bars](#)

**More information:**

[Updated on–line reference](#)



**LOWESTSINCEBARS****Lowest/Highest****– barssince lowest value since condition met**

(AFL 1.4)

**SYNTAX**     *lowestsincebars( **EXPRESSION**, **ARRAY**, **Nth = 1** )***RETURNS**     ARRAY**FUNCTION**     Returns the number of bars that have passed since lowest ARRAY value since EXPRESSION was true on the Nth most recent occurrence.**EXAMPLE**     lowestsincebars( Cross( macd(), 0 ), Close, 1 ) returns the number of bars passed since the lowest close price was detected from the time when macd() has crossed above zero.**SEE ALSO****References:**

The **LOWESTSINCEBARS** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**MA****Moving averages, summation****– simple moving average****SYNTAX**     *ma( ARRAY, periods)***RETURNS**     ARRAY**FUNCTION**     Calculates a *periods* simple moving average of ARRAY**EXAMPLE**     *ma(CLOSE, 5 )***SEE ALSO**     [TEMA\(\)](#) function , [AMA\(\)](#) function , [AMA2\(\)](#) function , [DEMA\(\)](#) function , [WMA\(\)](#) function , [WILDERS\(\)](#) function , [EMA\(\)](#) function**References:**

The **MA** function is used in the following formulas in AFL on–line library:

- [% B of Bollinger Bands With Adaptive Zones](#)
- ['DE\\*H37href='http://www.amibroker.com/library/detail.php?id=325target='\\_blank'>'R' Channel](#)
- [a](#)
- [abosliman2005m](#)
- [AC+ acceleration](#)
- [accum/dist mov avg crossover SAR](#)
- [AccuTrack](#)
- [ADXbuy](#)
- [AFL Example](#)
- [AFL Example – Enhanced](#)
- [AFL–Excel](#)
- [Against all odds](#)
- [Andrews PitchforkV3.3](#)
- [AO+ Momentum indicator](#)
- [Application of Ehler filter](#)
- [AR\\_Prediction.afl](#)
- [ATR Study](#)
- [Auto–Optimization Framework](#)
- [Balance of Power](#)
- [balance of power](#)
- [BB squeeze](#)
- [BMTRIX Intermediate Term Market Trend Indicator](#)
- [Bollinger – Keltner Bands](#)
- [Bollinger band normalization](#)
- [Bollinger Fibonacci Bands](#)
- [Bollinger oscillator](#)
- [Bow tie](#)
- [Breadth Thrust](#)
- [Bull/Bear Volume](#)
- [Bullish Percent Index 2 files combined](#)
- [CAMSLIM Cup and Handle Pattern AFL](#)
- [Candle Identification](#)
- [Candle Pattern Function](#)
- [Candle Stick Analysis](#)
- [Candlestick Volume Bars with Moving Average](#)

- CandleStochastics
- CCI Woodies Style
- CCI/DI+– COMBO indicator
- CCT Bollinger Band Oscillator
- CCT FibAccordion
- CCT Kaleidoscope
- Chaikin's Volatility
- Chande Momentum Oscillator
- Chande's Trend Score
- Chandelier Exit
- Chandelier Exit or Advanced Trailing Stop
- COMBO
- Compare Sectors against Tickers
- Coppock Curve
- crMathLib
- Cycle Highlighter
- Cycle Highlighter (auto best-fit)
- Dahl Oscillator modified
- danningham penetration
- Dave Landry PullBack Scan
- Dave Landry Pullbacks
- Demand Index
- Derivative Oscillator
- Divergences
- DMI Spread Index
- Double Smoothed Stochastic from W.Bressert
- DPO with shading
- DT Oscillator
- Dynamic Momentum Index
- Dynamic Momentum Index
- ekeko price chart
- Elder Bear Power
- Elder Bull Power
- Elder Impulse Indicator
- Elder Impulse Indicator V2
- Elder Ray – Bull Bear
- Elder Triple Screen Trading System.
- Elder's Market Thermometer
- Ema bands
- EMA Crossover
- EMA Crossover Price
- Ergodic Oscillator
- FastStochK FullStochK–D
- Fibonacci Moving averages
- fishnet
- Follow the Leader
- Force index
- Fund Screener
- Future MA Projection
- Futures – Dollar Move Today Indicator
- Gann Five Day pullback
- Gann HiLo Indicator and System

- garythompson
- garythompson
- Gordon Rose
- Guppy moving averages
- Hilbert Study
- Hull Moving Average
- Hurst "Like" DE
- IFT of RSI – Multiple TimeFrames
- INTRADAY HEIKIN ASHI new
- IntraDay Open Marker
- Keltner Channel
- Know Sure Thing
- Lagging MA–Xover
- MA Difference 20 Period
- MACD and histogram divergence detection
- MACD Histogram – Change in Direction
- Main price chart with Rainbow & SAR
- MAM
- Market Direction
- McClellan Oscillator
- McClellan Summation Index
- Momentum
- Momentum
- Moving Average "Crash" Test
- Moving Averages NoX
- MultiCycle 1.0
- nikhil
- Noor\_Doodie
- Pattern – Rectangle Base Breakout on High Vol
- Peterson
- Pivots for Intraday Forex Charts
- Plot Monthly, Weekly and Daily Moving average
- Polarized Fractal Efficiency
- prakash
- Price with Woodies Pivots
- Price–Volume Rank
- Projection Oscillator
- Pullback System No. 1
- QP2 Float Analysis
- QStick
- Rainbow Charts
- Rainbow Oscillator
- Relative Strength
- Relative strength comparison with moving average
- Reverse EMA function
- RSI Double–Bottom
- RSI of volume weighted moving average
- RSI Pointer
- RSIS
- RUTVOL timing signal with BB Scoring routine
- Sainath Sidgiddi
- Schiff Lines

- Sector Tracking
- SectorRSI
- SF Entry, Stop, PT Indicator
- SIROC Momentum
- Smoothed RSI Buy Signals
- STARC Bands
- STD\_STK Multi
- STO & MACD Buy Signals with Money-Management
- Stochastic Fast%K and Full
- Stochastic of Weekly Price Array
- StochD\_StochK Single.afl
- SunI>Support and Resistance
- Support Resistance levels
- T3
- T3 Function
- TAZ Trading Method Exploration
- The D\_oscillator
- The Mean RSI (variations)
- The Relative Slope
- The Relative Slope Pivots
- The Saturation Indicator D\_sat
- tomy\_frenchy
- Trend Detection
- Trend exploration with multiple timeframes
- Triangle exploration using PFChart
- Triangle search
- Triangular Moving Average
- Triangular Moving Average new
- Trigonometric Fit – TrigFit with AR for cos / sin
- TRIX
- TRIXXX
- Tushar Chande's Projected Range
- VAMA
- Varexlist
- Vertical Horizontal Filter (VHF)
- Vic Huebner
- Visualization of stoploses and profit in chart
- Volatility Breakout with Bollinger Bands
- Volatility Quality Index
- Volume – compared with Moving Avg (100%)
- Volume Oscillator
- Weekly Trend in Daily Graph
- Weinberg's The Range Indicator
- Williams %R with 9 period signal line
- Williams Alligator system
- ZeroLag MACD(p,q,r)
- \_Volume

**More information:**

Updated on-line reference



**MACD****Indicators****– moving average convergence/divergence****SYNTAX**     *macd(fast = 12, slow = 26)***RETURNS**     ARRAY**FUNCTION**     Calculates the MACD indicator using *fast* and *slow* averaging periods.**EXAMPLE**     The formula "macd()" returns the value of the MACD indicator (i.e., the red line). The formula "signal()" returns the value of the MACD's signal line (i.e., the blue line).**SEE ALSO**     The signal() function.**References:**

The **MACD** function is used in the following formulas in AFL on–line library:

- [Adaptave Zones O/B &O/S Oscillator](#)
- [AFL Example – Enhanced](#)
- [Bollinger band normalization](#)
- [Compare Sectors against Tickers](#)
- [Customised Avg. Profit %, Avg. Loss % etc](#)
- [Dinapoli Guru Commentary](#)
- [ekeko price chart](#)
- [Elder Impulse Indicator](#)
- [Elder Impulse Indicator V2](#)
- [Elder Triple Screen Trading System.](#)
- [Fund Screener](#)
- [hassan](#)
- [Indicator Explorer \(ZigZag\)](#)
- [MACD and histogram divergence detection](#)
- [MACD commentary](#)
- [MACD Histogram – Change in Direction](#)
- [MACD indicator display](#)
- [MACD optimize](#)
- [STO &MACD Buy Signals with Money–Management](#)
- [The Mean RSIt](#)
- [The Mean RSIt \(variations\)](#)
- [Trend Analysis\\_Comentary](#)
- [Trending or Trading](#)
- [Trending Ribbon](#)
- [ZeroLag MACD\(p,q,r\)](#)

**More information:**

[Updated on–line reference](#)

**MARKETID****Information / Categories****– market ID / name**

(AFL 1.8)

**SYNTAX**      *marketid( mode = 0 )***RETURNS**      NUMBER/STRING

**FUNCTION**      Retrieves current stock market ID/name When mode = 0 (the default value ) this function returns numerical marketID (consecutive market number)  
When mode = 1 this function returns name of the market.

**EXAMPLE**      Filter = MarketID() == 7 OR MarketID() == 9;  
AddTextColumn( MarketID( 1 ), "Market name" );

**SEE ALSO****References:**

The **MARKETID** function is used in the following formulas in AFL on–line library:

- [Alert Output As Quick Rewiev](#)
- [Auto–Optimization Framework](#)
- [Compare Sectors against Tickers](#)
- [Dave Landry PullBack Scan](#)
- [Elder Triple Screen Trading System.](#)

**More information:**

[Updated on–line reference](#)



**MAX****Math functions****– maximum value of two numbers / arrays****SYNTAX**     *max( ARRAY1, ARRAY2 )***RETURNS**     ARRAY**FUNCTION**     Returns the largest of the two parameters.**EXAMPLE**     The formula "max( CLOSE, 10 )" returns either the closing price or 10, whichever is greater.  
The formula "max(–14, 13)" always returns 13.**SEE ALSO****References:**The **MAX** function is used in the following formulas in AFL on–line library:

- ['D/9E H'](#)
- ['D/9E H'](#)
- ['R' Channel](#)
- [AR\\_Prediction.afl](#)
- [ATR Study](#)
- [Auto–Optimization Framework](#)
- [Bullish Percent Index 2 files combined](#)
- [Bullish Percent Index 2004](#)
- [CCI\(20\) Divergence Indicator](#)
- [Cycle Highlighter](#)
- [Cycle Highlighter \(auto best–fit\)](#)
- [Ed Seykota's TSP: EMA Crossover System](#)
- [Ed Seykota's TSP: Support and Resistance](#)
- [Ehler's filters and indicators](#)
- [EMA Crossover](#)
- [Gordon Rose](#)
- [INTRADAY HEIKIN ASHI new](#)
- [MACD and histogram divergence detection](#)
- [MACD indicator display](#)
- [PFChart – High/Low prices Sept2003](#)
- [PFchart with range box sizes](#)
- [Parabolic SAR in VBScript](#)
- [Performance Check](#)
- [PF Chart – Close – April 2004](#)
- [Pivots for Intraday Forex Charts](#)
- [Projection Oscillator](#)
- [Rainbow Oscillator](#)
- [Renko Chart](#)
- [Stops Implementation in AFS](#)
- [Three Line Break – TLB](#)
- [Trend Detection](#)
- [Triangle exploration using PFChart](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [Twiggs Money Flow](#)
- [Zig Zag](#)

**More information:**

[Updated on-line reference](#)

## **MDI** – minus directional movement indicator (–DI)

**Indicators**  
(AFL 1.3)

**SYNTAX**     *mdi( period = 14 )*

**RETURNS**     ARRAY

**FUNCTION**     Calculates Minus Directional Movement Indicator (–DI line)

**EXAMPLE**     mdi()

### **SEE ALSO**

#### **References:**

The **MDI** function is used in the following formulas in AFL on–line library:

- [ADX Indicator – Colored](#)
- [ADXbuy](#)
- [AJDX system](#)
- [CCI/DI+– COMBO indicator](#)
- [Dave Landry Pullbacks](#)
- [DMI Spread Index](#)
- [ekeko price chart](#)
- [Mndahoo ADX](#)
- [The Three Day Reversal](#)
- [Trend Analysis\\_Comentary](#)
- [Trending Ribbon](#)

#### **More information:**

[Updated on–line reference](#)

**MEDIAN****Statistical functions****– calculate median (middle element)**

(AFL 2.5)

**SYNTAX**     *Median( array, period )***RETURNS**     ARRAY**FUNCTION**     The Median function – finds median (middle element) value of the *array* over *period* elements.

**EXAMPLE**     *// list only symbols which volume is greater than*  
                  *// median Volume from past 50 days*  
                  **Filter** = **Volume** > **Median**( **Volume**, **50** );  
                  **AddColumn**( **V**, "Volume" );

**SEE ALSO****References:**

The **Median** function is used in the following formulas in AFL on–line library:

- [Adaptive Laguerre Filter, from John Ehlers](#)
- [AR\\_Prediction.afl](#)
- [Elder's Market Thermometer](#)
- [Noor\\_Doodie](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [Zig Zag](#)

**More information:**

[Updated on–line reference](#)

**MFI****Indicators****– money flow index**

**SYNTAX**      *mfi( periods = 14 )*

**RETURNS**     ARRAY

**FUNCTION**    Calculates the Money Flow Index with *period* range

**EXAMPLE**     mfi( 16 )

**SEE ALSO**     The rsi() function (see Relative Strength Index (RSI)).

**References:**

The **MFI** function is used in the following formulas in AFL on–line library:

- [Adaptave Zones O/B &O/S Oscillator](#)
- [Against all odds](#)
- [Bollinger band normalization](#)
- [DateNum\\_DateStr](#)
- [Ranking Ticker WatchList](#)
- [RSI "based" Trading System](#)
- [Smoothed RSI Buy Signals](#)
- [Volatility Breakout with Bollinger Bands](#)

**More information:**

[Updated on–line reference](#)

**MIN****Math functions****– minimum value of two numbers / arrays****SYNTAX**     *min( ARRAY1, ARRAY2 )***RETURNS**     ARRAY**FUNCTION**     Returns the smallest of the two parameters.**EXAMPLE**     The formula "min( CLOSE, 10 )" returns the closing price or 10, whichever is less. The formula "min(-14, 13)" always returns -14.**SEE ALSO**     The max() function.**References:**

The **MIN** function is used in the following formulas in AFL on-line library:

- ['D/9E H'](#)
- ['D/9E H'](#)
- ['R' Channel](#)
- [Andrews Pitchfork](#)
- [Andrews PitchforkV3.3](#)
- [AR\\_Prediction.afl](#)
- [ATR Study](#)
- [Bullish Percent Index 2 files combined](#)
- [Bullish Percent Index 2004](#)
- [CCI\(20\) Divergence Indicator](#)
- [Ed Seykota's TSP: Support and Resistance](#)
- [Elder Impulse Indicator V2](#)
- [EMA Crossover](#)
- [INTRADAY HEIKIN ASHI new](#)
- [MACD and histogram divergence detection](#)
- [MACD commentary](#)
- [MultiCycle 1.0](#)
- [PFChart – High/Low prices Sept2003](#)
- [PFchart with range box sizes](#)
- [Parabolic SAR in VBScript](#)
- [Performance Check](#)
- [PF Chart – Close – April 2004](#)
- [Pivots for Intraday Forex Charts](#)
- [Projection Oscillator](#)
- [Rainbow Oscillator](#)
- [Relative Strength Index](#)
- [Renko Chart](#)
- [Schiff Lines](#)
- [Stops Implementation in AFS](#)
- [Three Line Break – TLB](#)
- [Triangle exploration using PFChart](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [Twiggs Money Flow](#)
- [Weekly chart](#)

**More information:**

[Updated on-line reference](#)

**MINUTE****Date/Time****– get current bar's minute**

(AFL 2.0)

**SYNTAX**     *minute()***RETURNS**     ARRAY**FUNCTION**     Retrieves current bar's minute**EXAMPLE**     Hour()\*10000 + Minute() \* 100 + Second()**SEE ALSO**     Hour(), Second(), TimeNum()**References:**

The **MINUTE** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [Export Intraday Data](#)
- [Luna Phase](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**

[Updated on–line reference](#)



**MONTH**  
**– month****Date/Time**  
(AFL 1.4)**SYNTAX**     *month()***RETURNS**     ARRAY**FUNCTION**     Returns the array with months(1–12)**EXAMPLE**     buy = ( month() == 1 ) and day < 3; // buy in January**SEE ALSO****References:**

The **MONTH** function is used in the following formulas in AFL on–line library:

- [AB system full automation scripts](#)
- [Days to Third Friday](#)
- [Expiry Thursday for Indian markets](#)
- [Export Intraday Data](#)
- [Luna Phase](#)
- [LunarPhase](#)
- [Monthly bar chart](#)
- [N–period candlesticks \(time compression\)](#)
- [Option Calls, Puts and days till third friday.](#)
- [Relative Strength Multichart of up to 10 tickers](#)

**More information:**

[Updated on–line reference](#)

**MTRANDOM****Statistical functions****– Mersene Twister random number generator**

(AFL 3.0)

**SYNTAX**     *mtRandom( seed = Null )*  
                  *mtRandomA( seed = Null )*

**RETURNS**    NUMBER or ARRAY

**FUNCTION**    *mtRandom( seed = Null )* – returns single random number (scalar) in the range [0,1)  
                  *mtRandomA( seed = Null )* – returns array of random numbers in the range of [0,1)

seed is random generator seed value. If you don't specify one, the random number generator is automatically initialized with current time as a seed that guarantees unique sequence

Both functions use Mersene Twister mt19973ar-cok algorithm. (Copyright (C) 1997 – 2002, Makoto Matsumoto and Takuji Nishimura.)

Mersene Twister is vastly superior to C-runtime pseudo-random generator available via Random() function.

It has a period of  $2^{19973}$  = approx  $2.9 \cdot 10^{6012}$  For more information visit:  
<http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>

See also: M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator", ACM Transactions on Modeling and Computer Simulation, Vol. 8, No. 1, January 1998, pp 3—30.

**EXAMPLE**     `printf("Random number: %g", mtRandom() );`

**SEE ALSO**     [RANDOM\(\)](#) function

**References:**

The **mtRandom** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)

**NAME****Information / Categories****– ticker symbol**

(AFL 1.1)

**SYNTAX**     *name()***RETURNS**     STRING**FUNCTION**     It is used to display the stock short name (ticker)**EXAMPLE**     name()**SEE ALSO****References:**

The **NAME** function is used in the following formulas in AFL on–line library:

- 'D/9E H'
- 'D/9E H'
- 3 Price Break
- AccuTrack
- ADX Indicator – Colored
- AFL Example
- AFL Example – Enhanced
- AFL–Excel
- Against all odds
- Alert Output As Quick Rewiev
- Alpha and Beta and R\_Squared Indicator
- Aroon Indicators
- Auto–Optimization Framework
- Baseline Relative Performance Watchlist charts V2
- Bollinger – Keltner Bands
- Bottom Trader
- Bull Fear / Bear Fear
- Bullish Percent Index 2004
- CCI Woodies Style
- CCI(20) Divergence Indicator
- Cole
- Color Coded Short Term Reversal Signals
- colored CCI
- Commodity Channel Index
- Dave Landry PullBack Scan
- De Mark's Range Projection
- Dinapoli Guru Commentary
- Double Smoothed Stochastic from W.Bressert
- Double top detection
- DPO with shading
- Elder Triple Screen Trading System.
- Elder's Market Thermometer
- Elder's SafeZone Stop
- ElderSafeZoneStopLong
- ElderSafeZoneStopShort
- EMA Crossover Price

- Export Intraday Data
- FastStochK FullStochK-D
- Fibonacci Moving averages
- Futures – Dollar Move Indicator
- Futures – Dollar Move Today Indicator
- Gordon Rose
- hassan
- Indicator Explorer (ZigZag)
- INTRADAY HEIKIN ASHI new
- Larry William's Volatility Channels
- MA Difference 20 Period
- MACD and histogram divergence detection
- MACD commentary
- MACD Histogram – Change in Direction
- Main price chart with Rainbow & SAR
- Mndahoo ADX
- Monthly Coppock Guide
- Moving Averages NoX
- Noor\_Doodie
- PFChart – High/Low prices Sept2003
- Parabolic SAR in JScript
- Parabolic SAR in VBScript
- ParabXO
- Performance Overview
- Peterson
- PF Chart – Close – April 2004
- Pivot Finder
- Pivot Point and Support and Resistance Points
- Pivots for Intraday Forex Charts
- Plot Monthly, Weekly and Daily Moving average
- Position Sizing and Risk Price Graph
- Position Sizing and Risk Price Graph – 2
- prakash
- Price with Woodies Pivots
- Probability Calculator
- QP2 Float Analysis
- Rainbow Charts
- Rainbow Oscillator
- Ranking Ticker WatchList
- Rea Time Daily Price Levels
- Relative Strength
- Relative Strength Index
- Renko Chart
- ROC of MACD Weekly
- RSI of Weekly Price Array
- RSIS
- Shares To Buy Price Graph
- Simple Momentum
- STD\_STK Multi
- Steve Woods' Cum. Vol. Float + Cum. Vol. Channels
- Steve Woods' Cumulative Vol. Percentage Indicator
- Steve Woods' Float Channel Lines

- Stochastic Fast%K and Full
- Stochastics Trendlines
- StochD\_StochK Single.afl
- Stock price AlertIf
- SunI>Support Resistance levels
- The D\_oscillator
- Three Line Break – TLB
- Triangular Moving Average
- Triangular Moving Average new
- TRIX
- Twiggs Money Flow
- Ultimate plus
- UltraEdit editor highlight wordfile
- Weinberg's The Range Indicator
- William's % R
- Williams %R with 9 period signal line
- Williams %R Exploration
- Woodies CCI
- Zig Zag Indicator with Valid Entry and Exit Points
- ZigZag filter rewritten from scratch in AFL
- \_Volume

**More information:**

[Updated on-line reference](#)

**NOTEGET****– retrieves the text of the note****Miscellaneous functions**  
(AFL 2.6)**SYNTAX**     *NoteGet( "Symbol" )***RETURNS**    STRING**FUNCTION**    Retrieves note linked to "symbol". If symbol is "" (empty string) then current symbol is used.  
If symbol is "" (empty string) then current symbol is used.**EXAMPLE**     "You have entered the following text in the notepad" + *NoteGet( "" )* ;**SEE ALSO**     *NoteSet()* function**References:**

The **NoteGet** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**NOTESET****Miscellaneous functions**  
(AFL 2.6)**– sets text of the note****SYNTAX**     *NoteSet*( "Symbol", "Text.." );**RETURNS**     NUMBER**FUNCTION**     Sets text of the note linked to "symbol".  
If symbol is "" (empty string) then current symbol is used.

If you overwrite note from AFL level that is opened at the same time in Notepad editor the editor will ask you (when you switch the focus to it) if it should reload new text or allow to save your manually entered text.

Returns True (1) on success, 0 on failure.

**EXAMPLE**     `NoteSet( "AMD", "Jun 15, 2004: AMD will deliver its first multi-core processors next year" );`**SEE ALSO**     `NoteGet()` function**References:**

The **NoteSet** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**NOW****Date/Time**  
(AFL 2.3)**– gets current system date/time****SYNTAX**     *Now( format = 0 )***RETURNS**     STRING or NUMBER**FUNCTION**     Returns current date / time in numerous of formats:

- format = 0 – returns string containing current date/time formatted according to system settings
- format = 1 – returns string containing current date only formatted according to system settings
- format = 2 – returns string containing current time only formatted according to system settings
- format = 3 – returns DATENUM number with current date
- format = 4 – returns TIMENUM number with current time
- format = 5 – returns DATETIME number with current date/time
- format = 6 – returns current DAY (1..31)
- format = 7 – returns current MONTH (1..12)
- format = 8 – returns current YEAR (four digit)
- format = 9 – returns current DAY OF WEEK (1..7, where 1=Sunday, 2=Monday, and so on)
- format = 10 – returns current DAY OF YEAR (1..366)

**EXAMPLE**     AddTextColumn( Now(), "Current time");**SEE ALSO**     [DATENUM\(\)](#) function , [DATETIME\(\)](#) function , [DATE\(\)](#) function , [TIMENUM\(\)](#) function**References:**The **NOW** function is used in the following formulas in AFL on–line library:

- [AFL Timing functions](#)
- [CCI\(20\) Divergence Indicator](#)
- [Time Left in Bar](#)
- [TWS auto–export Executions–file parser](#)

**More information:**[Updated on–line reference](#)



**NUMTOSTR****String manipulation****– convert number to string**

(AFL 2.5)

**SYNTAX**      *NumToStr( NUMBER, format = 1.3, separator=True)*  
                  *NumToStr( ARRAY, format = 1.3, separator=True )*

**RETURNS**    STRING

**FUNCTION**    It is used to convert numeric value of NUMBER or ARRAY to string.

The second parameter – *format* – allows you to control output formatting (decimal places and leading spaces). The integer part of the number controls minimum number of characters used to display the number (if you specify high number the output will be space-padded). The fractional part defines how many decimal places to display, for example 1.0 – will give you a number without fractional part at all, and 1.2 – will give two digits past the decimal point

There is also a special format constant `formatDateTime` that allows to convert date/time returned by `DateTime()` function formatted according to Windows regional settings.

Third parameter *separator* (true by default) controls if thousand separator is added or not. Thousands separator is definable in Tools->Preferences->Misc.

Note: **NumToStr** is a synonym for **WriteVal** function.

**EXAMPLE**    1. Simple use (no custom format)

```
NumToStr( StochK( 39 ) - StochK( 12 ) );
```

2. Display rate of change with 2 decimal digits and % appened to the end

```
NumToStr( ROC( Close, 20 ), 1.2 ) + "%";
```

3. Display date/time according to regional settings

```
NumToStr( DateTime(), formatDateTime );
```

**SEE ALSO**    `WRITEVAL()` function , `StrToNum()` function

**References:**

The **NumToStr** function is used in the following formulas in AFL on-line library:

- 'D/9E H'
- 'D/9E H'
- 3 Price Break
- AFL Timing functions
- AR\_Prediction.afl
- Candle Identification
- Candle Stick Analysis
- Candle Stick Demo
- Cycle Highlighter
- Cycle Highlighter (auto best-fit)

- [DateNum\\_DateStr](#)
- [Fibonacci Moving averages](#)
- [Gordon Rose](#)
- [Hurst "Like" DE](#)
- [lastNDaysBeforeDate](#)
- [nth \( 1 – 8 \) Order Polynomial Fit](#)
- [Option Calls, Puts and days till third friday.](#)
- [prakash](#)
- [Ranking and sorting stocks](#)
- [Time Left in Bar](#)
- [tomy\\_frenchy](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [TWS auto-export Executions-file parser](#)
- [WLBUILDProcess](#)

**More information:**

[Updated on-line reference](#)

**NVI****Indicators****– negative volume index****SYNTAX**     *nvi()***RETURNS**     ARRAY**FUNCTION**     Calculates the Negative Volume Index.**EXAMPLE****SEE ALSO**     The pvi() function**References:**

The **NVI** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**NZ****Miscellaneous functions**  
(AFL 2.3)**– Null (Null/Nan/Infinity) to zero****SYNTAX**     **Nz( x, valueifnull = 0 )****RETURNS**     NUMBER, ARRAY**FUNCTION**     Converts Null/Nan/Infinity values to zero (or user defined value)

x can be number or array.

You can use the Nz function to return zero, or another specified value when argument x is Null or Nan or Infinite.

For example, you can use this function to convert a Null (empty) value to another value and prevent it from propagating through an expression. If the optional valueifnull argument is included, then the Nz function will return the value specified by that argument if the x argument is Null (or Nan or Infinity).

**EXAMPLE**     You can use the Nz function as an alternative to the IIf function.

Instead of:

```
varTemp = IIf( IsFinite( (H-L)/(C-L) ), (H-L)/(C-L), 0 );
```

You can write:

```
varTemp = Nz( (H-L)/(C-L) );
```

**SEE ALSO****References:**

The **NZ** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [Candle Stick Analysis](#)
- [Dynamic Momentum Index](#)
- [Dynamic Momentum Index](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**

[Updated on–line reference](#)

**OBV****Indicators****– on balance volume****SYNTAX**     *obv()***RETURNS**     ARRAY**FUNCTION**     Calculates the On Balance Volume indicator.**EXAMPLE****SEE ALSO****References:**

The **OBV** function is used in the following formulas in AFL on–line library:

- [RSI "based" Trading System](#)
- [Smoothed RSI Buy Signals](#)
- [Stochastic OBV and Price Filter](#)

**More information:**

[Updated on–line reference](#)

**OPTIMIZE**

Trading system toolbox

– define optimization variable

(AFL 1.7)

**SYNTAX**     *optimize( "description", default, min , max, step )***RETURNS**     NUMBER

**FUNCTION**     Defines optimization process parameters. With normal backtesting, scanning, exploration and comentary modes the optimize function returns *default* value, so the above function call returns *default*; In optimization mode optimize function returns successive values from *min* to *max* (inclusively) with *step* stepping. "*description*" is a string that is used to identify the optimization variable and is displayed as a column name in the optimization result list. *default* is a default value that optimize function returns in exploration, indicator, commentary, scan and normal back test modes *min* is a minimum value of the variable being optimized *max* is a maximum value of the variable being optimized *step* is an interval used for increasing the value from *min* to *max*

**EXAMPLE**     variable = optimize("my optimization var", 9, 2, 20, 1 );

**SEE ALSO****Comments:**

<b>Herman van den Bergen</b> psytek@magma.ca 2003-06-09 05:23:31	You can Optimize parameters with custom number series by using the numbers generated by the Optimize() function as an index to access numbers in a custom array. Here is an example using a custom array FB[] of Fibonacci numbers:  FB[0] = 0.0; FB[1] = 23.6; FB[2] = 38.2; FB[3] = 50.0; FB[4] = 61.8; FB[5] = 100; FB[6] = 161.8; FB[7] = 261.8; FB[8] = 423.6; FBIndex = Optimize("FBIndex",0,0,8,1); FibNum = FB[FBIndex]; ... place your Code using FibNum here ...
<b>Herman van den Bergen</b> psytek@magma.ca 2003-07-20 17:26:08	You can refresh your Equity chart after each Optimization step and observe (like a slide show) how the linearity of your Equity curve is effected by adding these two lines to the very end of your code:  AB = CreateObject("Broker.Application"); AB.RefreshAll();  Important note: Do not use in commentary, interpretation, or indicator builder because it will cause loop. (Thanks for the tip TJ!)
<b>Graham Kavanagh</b> gkavanagh@e-wire.net.au 2004-08-21 23:31:39	When optimising for 2 or more variables make sure you have different names for each variable. eg x = Optimize("Short",5,5,10,1); y = Optimize("Short",15,25,55,1);  I made mistake of copy/paste and did not change the optimize name (as above) within the brackets and got all zeroes as results.

	<p>This below gets results</p> <pre>x = Optimize("Short",5,5,10,1); y = Optimize("Long",15,25,55,1);</pre> <p>Graham</p>
<p><b>Tomasz Janeczko</b>  tj --at-- amibroker.com  2006-12-12 11:30:18</p>	<p>Some asked for function that combines Param() and Optimize(). Here it is:</p> <pre>function ParamOptimize( pname, defaultval, minv, maxv, step ) { return Optimize( pname, Param( pname, defaultval, minv, maxv, step ), minv, maxv, step ); }</pre>

**References:**

The **OPTIMIZE** function is used in the following formulas in AFL on-line library:

- [ADXbuy](#)
- [ATR Study](#)
- [Auto-Optimization Framework](#)
- [Bollinger band normalization](#)
- [Bull Fear / Bear Fear](#)
- [Dahl Oscillator modified](#)
- [danningham penetration](#)
- [Ed Seykota's TSP: EMA Crossover System](#)
- [Ed Seykota's TSP: Support and Resistance](#)
- [eeko price chart](#)
- [EMA Crossover](#)
- [Evaluating Candle Patterns in a trading system](#)
- [FastStochK FullStochK-D](#)
- [Fund Screener](#)
- [Lagging MA-Xover](#)
- [MACD and histogram divergence detection](#)
- [MACD optimize](#)
- [Modified Darvas Box](#)
- [Moving Averages NoX](#)
- [OptimizationBatch.js](#)
- [Peterson](#)
- [Projection Oscillator](#)
- [RUTVOL timing signal with BB Scoring routine](#)
- [SectorRSI](#)
- [STD\\_STK Multi](#)
- [Stochastic Fast%K and Full](#)
- [Stochastic optimize](#)
- [StochD\\_StochK Single.afl](#)
- [The D\\_oscillator](#)
- [Trend Continuation Factor](#)
- [Trend Trigger Factor](#)
- [TRIX](#)
- [Volatility System](#)

**More information:**

[Updated on-line reference](#)



**OSCP****Indicators****– price oscillator****SYNTAX**      *oscp( fast , slow )***RETURNS**     ARRAY**FUNCTION**     Calculates price oscillator based on exponential moving averages**EXAMPLE**      oscp(9, 18)**SEE ALSO****References:**

The **OSCP** function is used in the following formulas in AFL on–line library:

- [Indicator Explorer \(ZigZag\)](#)
- [MACD commentary](#)

**More information:**

[Updated on–line reference](#)

**OSCV****Indicators****– volume oscillator****SYNTAX**     **oscv( *fast, slow* )****RETURNS**     ARRAY**FUNCTION**     Calculates volume oscillator based on exponential moving averages**EXAMPLE**     oscv( 9, 18 )**SEE ALSO****References:**

The **OSCV** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**OUTSIDE**

Basic price pattern detection

**– outside bar****SYNTAX**     *outside()***RETURNS**     ARRAY**FUNCTION**     Gives "true" (or 1) when an outside day occurs**EXAMPLE**     *outside()***SEE ALSO****References:**

The **OUTSIDE** function is used in the following formulas in AFL on–line library:

- [AC+ acceleration](#)
- [AO+ Momentum indicator](#)
- [Fund Screener](#)
- [Gann Swing Chart](#)
- [Vic Huebner](#)
- [Williams Alligator system](#)

**More information:**

[Updated on–line reference](#)

**PARAM****Exploration / Indicators****– add user user–definable numeric parameter**

(AFL 2.3)

**SYNTAX**     *Param( "name", defaultval, min, max, step, sincr = 0 )***RETURNS**     NUMBER**FUNCTION**     Adds a new user–definable parameter, which will be accessible via Parameters dialog : right click over chart pane and select "Parameters" or press Ctrl+R allows to change chart parameters – changes are reflected immediately.

- "name" – defines parameter name that will be displayed in the parameters dialog
- defaultval – defines default value of the parameter
- min, max – define minimum and maximum values of the parameter
- step – defines minimum increase of the parameter via slider in the Parameters dialog
- sincr – automatic section increment value (used by drag–drop interface to increase default values for parameters)

**WARNING:** default/min/max/step parameters have to be CONSTANT numbers. This is because these values are cached and are not re–read during subsequent formula evaluations.

**EXAMPLE**     Sample code 1:

```
ticker = ParamStr( "Ticker", "MSFT" );
sp = Param( "MA Period", 12, 2, 100 );
PlotForeign( ticker, "Chart of "+ticker, ParamColor( "Price Color",
colorLightYellow ), styleCandle );
Plot( MA( Foreign( ticker, "C" ), sp ), "MA(" + WriteVal( sp, 1.0 )
+ ")", ParamColor( "MA Color", colorRed ) );
```

Sample code 2:

```
sp = Param( "RSI Period", 12, 2, 100 );
r = RSI( sp );
Plot( r, "RSI("+WriteVal(sp,1.0)+")", ParamColor("RSI Color",
colorRed ) );

Buy = Cross( r, 30 );
Sell = Cross( 70, r );

PlotShapes( shapeUpArrow * Buy + shapeDownArrow * Sell, IIf( Buy,
colorGreen, colorRed ) );
```

**SEE ALSO**     [PARAMCOLOR\(\)](#) function , [PARAMSTR\(\)](#) function [PARAMCOLOR\(\)](#) function , [ParamTime\(\)](#) function , [ParamDate\(\)](#) function

**Comments:**

<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2006–02–19	Note that Parameters are INDEPENDENT for each chart pane and for Automatic Analysis window. In Automatic Analysis window parameters can be modified using "Parameters" button and they are independent from ones you use for any chart.
--	--

06:18:23	
<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2006-02-19 06:19:59	To change the parameters for the indicator, please click with RIGHT mouse button over chart pane and select "Parameters" from the menu.

**References:**

The **PARAM** function is used in the following formulas in AFL on-line library:

- 'D/9E H'
- 'D/9E H'
- 'DE\*H3href='http://www.amibroker.com/library/detail.php?id=413target='\_blank'>3 Price Break
- abosliman2005m
- Adaptive Laguerre Filter, from John Ehlers
- Adaptive Price Channel
- ADX Indicator – Colored
- AFL Example
- AFL Example – Enhanced
- AFL Timing functions
- An n bar Reversal Indicator
- Andrews Pitchfork
- Andrews PitchforkV3.3
- Application of Ehler filter
- AR\_Prediction.afl
- Baseline Relative Performance Watchlist charts V2
- Black Scholes Option Pricing
- Bollinger – Keltner Bands
- Bollinger Fibonacci Bands
- Bottom Trader
- Buff Volume Weighted Moving Averages
- Bull/Bear Volume
- Candle Stick Analysis
- Candle Stick Demo
- CCI(20) Divergence Indicator
- Chandelier Exit
- Chandelier Exit or Advanced Trailing Stop
- Cycle Highlighter
- Cycle Highlighter (auto best-fit)
- Dave Landry PullBack Scan
- DPO with shading
- DT Oscillator
- Dynamic Momentum Index
- Dynamic Momentum Index
- Ehlers Center of Gravity Oscillator
- ekeko price chart
- EKEKO SAR-MF
- Elder Impulse Indicator
- Elder Impulse Indicator V2
- Elder safe Zone Long + short
- Elder Triple Screen Trading System.

- Elder's Market Thermometer
- Elder's SafeZone Stop
- ElderSafeZoneStopLong
- ElderSafeZoneStopShort
- Ergodic Oscillator
- FastStochK FullStochK-D
- Fib CMO
- Fibonacci Moving averages
- Future MA Projection
- Gordon Rose
- Hull Moving Average
- Hurst "Like" DE
- Indicator Explorer (ZigZag)
- INTRADAY HEIKIN ASHI new
- Larry William's Volatility Channels
- Linear Regression Line w/ Std Deviation Channels
- Log Time Scale
- LunarPhase
- MACD and histogram divergence detection
- MACD Histogram – Change in Direction
- MACD indicator display
- Market Profile & Market Volume Profile
- Mndahoo ADX
- Monte Carlo
- Moving Averages NoX
- MultiCycle 1.0
- Multiple sinus noised
- Noor\_Doodie
- nth ( 1 – 8 ) Order Polynomial Fit
- Option Calls, Puts and days till third friday.
- ParabXO
- Pivot Finder
- Pivot Point with S/R Trendlines
- Position Sizing and Risk Price Graph – 2
- prakash
- Projection Oscillator
- Random Walk
- Ranking and sorting stocks
- Raw ADX
- Rea Time Daily Price Levels
- Relative strength comparison with moving average
- Renko Chart
- RSI of volume weighted moving average
- SAR-ForNextBarStop
- Schiff Lines
- Shares To Buy Price Graph
- Simple Momentum
- Standard Error Bands (Native AFL)
- STARC Bands
- SUPER PIVOT POINTS
- Support and Resistance
- Support Resistance levels

- TD sequential
- The Fibonacci behavior
- The Three Day Reversal
- tomy\_frenchy
- Trend Detection
- Triangular Moving Average new
- Trigonometric Fit – TrigFit with AR for cos / sin
- Twiggs Money Flow
- Ultimate plus
- VAMA
- Visualization of stoploses and profit in chart
- William's % R
- Williams %R Exploration
- Woodies CCI
- ZeroLag MACD(p,q,r)
- Zig Explorer
- Zig Zag
- ZigZag filter rewritten from scratch in AFL
- \_Volume

**More information:**

Updated on-line reference

**PARAMCOLOR****Exploration / Indicators****– add user user–definable color parameter**

(AFL 2.3)

**SYNTAX**     *ParamColor( "name", defaultcolor )***RETURNS**     NUMBER

**FUNCTION**     Adds a new user–definable parameter, which will be accessible via Parameters dialog : right click over chart pane and select "Parameters" or press Ctrl+R allows to change chart parameters – changes are reflected immediately.

- "name" – defines parameter name that will be displayed in the parameters dialog
- defaultcolor – defines default color value of the parameter

colorCycle – accepted only by ParamColor function as default value, causes that default color cycles through red, blue, green, turquoise, gold, violet, bright green, dark yellow

**EXAMPLE**     `Plot( RSI(), "RSI", ParamColor( "RSI Color", colorRed ) );`

**SEE ALSO**     [PARAM\(\)](#) function , [PARAMSTR\(\)](#) function

**References:**

The **ParamColor** function is used in the following formulas in AFL on–line library:

- [Adaptive Laguerre Filter, from John Ehlers](#)
- [Andrews Pitchfork](#)
- [Andrews PitchforkV3.3](#)
- [AR\\_Prediction.afl](#)
- [Chandelier Exit](#)
- [Cycle Highlighter](#)
- [Cycle Highlighter \(auto best–fit\)](#)
- [Dave Landry PullBack Scan](#)
- [DT Oscillator](#)
- [Elder safe Zone Long + short](#)
- [Elder Triple Screen Trading System.](#)
- [FastStochK FullStochK–D](#)
- [Fib CMO](#)
- [Fibonacci Moving averages](#)
- [INTRADAY HEIKIN ASHI new](#)
- [IntraDay Open Marker](#)
- [Linear Regression Line w/ Std Deviation Channels](#)
- [LunarPhase](#)
- [Mndahoo ADX](#)
- [Multiple sinus noised](#)
- [nifty](#)
- [Option Calls, Puts and days till third friday.](#)
- [ParabXO](#)
- [Pivot Point with S/R Trendlines](#)
- [prakash](#)
- [Random Walk](#)
- [SAR–ForNextBarStop](#)
- [Schiff Lines](#)



- [Simple Momentum](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [Twiggs Money Flow](#)
- [Ultimate plus](#)
- [VAMA](#)
- [William's % R](#)
- [Williams %R Exploration](#)
- [Woodies CCI](#)
- [ZigZag filter rewritten from scratch in AFL](#)
- [\\_Volume](#)

**More information:**

[Updated on-line reference](#)

**PARAMDATE****Exploration / Indicators****– add user user–definable date parameter**

(AFL 2.60)

**SYNTAX**     *ParamDate( "Name", "Default date", format = 0 );***RETURNS**     NUMBER or STRING**FUNCTION**     Adds a new user–definable date parameter, which will be accessible via Parameters dialog : right click over chart pane and select "Parameters" or press Ctrl+R allows to change chart parameters – changes are reflected immediately.

- "name" – defines parameter name that will be displayed in the parameters dialog
- "default date" – is a string holding date in any any format: YYYY–MM–DD, MM/DD/YY, DD–MM–YY, etc, etc.
- format – defines return value format, allowable values are:
  - 0 – return value is a NUMBER and holds DateNum. Ie: 990503 for May 3, 1999,
  - 1 – return value is a STRING formatted holding date according to your windows regional settings

**WARNING:** default parameter has to be CONSTANT. This is because these values are cached and are not re–read during subsequent formula evaluations.

**EXAMPLE**     `start = ParamDate( "Start Date", "2003-05-03" );`**SEE ALSO**     [PARAM\(\)](#) function , [PARAMCOLOR\(\)](#) function , [PARAMSTR\(\)](#) function , [ParamTime\(\)](#) function**References:**

The **ParamDate** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**PARAMFIELD****Exploration / Indicators****– creates price field parameter**

(AFL 2.70)

**SYNTAX**     *ParamField("name", field = 3 )***RETURNS**     ARRAY

**FUNCTION**     Allows to pick the **Price field** for the indicator (field which is used to calculate values of the indicator). Function returns the array defined by *field* parameter. Default value = 3 returns Close array. The possible values of *field* parameter are:

- **-1** – ParamField returns the values of the indicator that was inserted as a first one into the pane, or Close if no indicator was present
- **0** – returns **Open** array
- **1** – returns **High** array
- **2** – returns **Low** array
- **3** – returns **Close** array (default)
- **4** – returns **Average** array = (H+L+C)/3
- **5** – returns **Volume** array
- **6** – returns **Open Interest** array
- **7,8,9,....** – return values of indicators inserted into the pane.

**EXAMPLE****SEE ALSO**     [PARAM\(\)](#) function**References:**

The **ParamField** function is used in the following formulas in AFL on–line library:

- [Adaptive Laguerre Filter, from John Ehlers](#)
- [AR\\_Prediction.afl](#)
- [Bottom Trader](#)
- [Dave Landry PullBack Scan](#)
- [Elder Triple Screen Trading System.](#)
- [Elder's Market Thermometer](#)
- [Fibonacci Moving averages](#)
- [INTRADAY HEIKIN ASHI new](#)
- [Linear Regression Line w/ Std Deviation Channels](#)
- [Noor\\_Doodie](#)
- [prakash](#)
- [tomy\\_frenchy](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [\\_Volume](#)

**More information:**

[Updated on–line reference](#)

**PARAMLIST****Exploration / Indicators****– creates the parameter that consist of the list of choices**

(AFL 2.70)

**SYNTAX**     *ParamList( "Name", "Values", defaultval = 0 )***RETURNS**     STRING

**FUNCTION**     Creates the parameter that consist of the list of choices (specified in "values" parameter – | or comma separated). *defaultval* parameter defines ordinal position of the default string value specified in "values" parameter. Returned value is a STRING representing choosen item.

**EXAMPLE**     `OrderType = ParamList( "Order Type", "MKT|LMT|STP" );`

**SEE ALSO**     [ParamDate\(\)](#) function , [PARAMSTR\(\)](#) function , [ParamTime\(\)](#) function , [ParamTrigger\(\)](#) function , [PARAMCOLOR\(\)](#) function , [PARAM\(\)](#) function

**References:**

The **ParamList** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [Hurst "Like" DE](#)
- [SUPER PIVOT POINTS](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**

[Updated on–line reference](#)

**PARAMSTR****Exploration / Indicators****– add user user–definable string parameter**

(AFL 2.3)

**SYNTAX**     *ParamStr( "name", "default" )***RETURNS**    STRING

**FUNCTION**    Adds a new user–definable parameter, which will be accessible via Parameters dialog : right click over chart pane and select "Parameters" or press Ctrl+R allows to change chart parameters – changes are reflected immediately.

- "name" – defines parameter name that will be displayed in the parameters dialog
- "default" – defines default value of the parameter

**EXAMPLE**     `ticker = ParamStr( "Ticker", "MSFT" );`

**SEE ALSO**     [PARAM\(\)](#) function , [PARAMCOLOR\(\)](#) function

**References:**

The **PARAMSTR** function is used in the following formulas in AFL on–line library:

- [Relative Strength Multichart of up to 10 tickers](#)

**More information:**

[Updated on–line reference](#)

**PARAMSTYLE****Exploration / Indicators****– select styles applied to the plot**

(AFL 2.70)

**SYNTAX**      *ParamStyle("name", defaultval = styleLine, mask = maskDefault )***RETURNS**      NUMBER**FUNCTION**      Allows to select the styles applied to plot.

## Parameters

- name – parameter name
- defaultval – default value of style , takes combination of style\* constants
- mask – binary mask that defines which styles should be visible in the drop down list  
maskDefault – show thick, dashed, hidden, own scale styles (this is default mask for ParamStyle)  
maskAll – show all style flags  
maskPrice – show thick, hidden, own scale, candle, bar  
maskHistogram – show histogram, thick, hidden, own scale, area

**EXAMPLE****SEE ALSO****References:**

The **ParamStyle** function is used in the following formulas in AFL on–line library:

- [Adaptive Laguerre Filter, from John Ehlers](#)
- [AR\\_Prediction.afl](#)
- [Chandelier Exit](#)
- [Cycle Highlighter](#)
- [Cycle Highlighter \(auto best–fit\)](#)
- [Dave Landry PullBack Scan](#)
- [Elder safe Zone Long + short](#)
- [Elder Triple Screen Trading System.](#)
- [FastStochK FullStochK–D](#)
- [Fibonacci Moving averages](#)
- [INTRADAY HEIKIN ASHI new](#)
- [Linear Regression Line w/ Std Deviation Channels](#)
- [LunarPhase](#)
- [Market Profile &Market Volume Profile](#)
- [Mndahoo ADX](#)
- [Multiple sinus noised](#)
- [nifty](#)
- [Option Calls, Puts and days till third friday.](#)
- [prakash](#)
- [Random Walk](#)
- [SAR–ForNextBarStop](#)
- [Simple Momentum](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [Twiggs Money Flow](#)
- [Ultimate plus](#)

- [ZigZag filter rewritten from scratch in AFL](#)
- [\\_Volume](#)

**More information:**

[Updated on-line reference](#)

**PARAMTIME****Exploration / Indicators****– add user user–definable time parameter**

(AFL 2.60)

**SYNTAX**     *ParamTime( "Name", "Default time", format = 0 );***RETURNS**     NUMBER or STRING**FUNCTION**     Adds a new user–definable time parameter, which will be accessible via Parameters dialog : right click over chart pane and select "Parameters" or press Ctrl+R allows to change chart parameters – changes are reflected immediately.

- "name" – defines parameter name that will be displayed in the parameters dialog
- "default time" – is a string holding time in any any format: HH:MM:SS, HH:MM, etc.
- format – defines return value format, allowable values are:
  - 0 – return value is a NUMBER and holds TimeNum. Ie: 133515 for 13:35:15
  - 1 – return value is a STRING formatted holding time according to your windows regional settings

**WARNING:** default parameter has to be CONSTANT. This is because these values are cached and are not re–read during subsequent formula evaluations.

**EXAMPLE**     `start = ParamTime( "Start Time", "09:30" );`**SEE ALSO**     [PARAM\(\)](#) function , [PARAMCOLOR\(\)](#) function , [ParamDate\(\)](#) function , [PARAMSTR\(\)](#) function**References:**

The **ParamTime** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)



**PARAMTOGGLE****Exploration / Indicators****– create Yes/No parameter**

(AFL 2.70)

**SYNTAX**     *ParamToggle("name","values",defaultval =0 )***RETURNS**     NUMBER**FUNCTION**     Function that allows to use boolean (Yes/No) parameters.

- "name" – the name of the parameter
- "values" – parameter values (separated with | character, e.g. "No|Yes" – first string represents false value and second string represents true value)
- defaultval – default value of the parameter

**EXAMPLE****SEE ALSO**     [PARAM\(\)](#) function , [ParamTrigger\(\)](#) function**References:**

The **ParamToggle** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [Cycle Highlighter](#)
- [Cycle Highlighter \(auto best–fit\)](#)
- [Evaluating Candle Patterns in a trading system](#)
- [Fibonacci Moving averages](#)
- [Market Profile &Market Volume Profile](#)
- [Monte Carlo](#)
- [Multiple sinus noised](#)
- [prakash](#)
- [Random Walk](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [Visualization of stoploses and profit in chart](#)

**More information:**

[Updated on–line reference](#)

**PARAMTRIGGER****Exploration / Indicators****– creates a trigger (button) in the parameter dialog**

(AFL 2.70)

**SYNTAX**     *ParamTrigger( "Name", "Button text")***RETURNS**    NUMBER**FUNCTION**    Creates trigger (button) in the Parameter dialog.

If you place ParamTrigger in the indicator code it will create a "button" in Parameter dialog that can be pressed. Normally ParamTrigger will return zero (0) but when button in the param window is pressed then it will refresh the chart and ParamTrigger will return 1 (one) for this single execution (further refreshes will return zero, until the button is pressed again)

**EXAMPLE**     `trigger = ParamTrigger("Place Order", "Click here to place order");`

```
if( trigger )
{
// your one-shot code here
}
```

**SEE ALSO****References:**

The **ParamTrigger** function is used in the following formulas in AFL on–line library:

- [AFL Timing functions](#)

**More information:**

[Updated on–line reference](#)

## **PDI** – plus directional movement indicator

**Indicators**  
(AFL 1.3)

**SYNTAX**     *pdi( period = 14 )*

**RETURNS**     ARRAY

**FUNCTION**     Calculates Plus Directional Movement Indicator (–DI line)

**EXAMPLE**     decvolume()

### **SEE ALSO**

#### **References:**

The **PDI** function is used in the following formulas in AFL on–line library:

- [ADX Indicator – Colored](#)
- [ADXbuy](#)
- [AJDX system](#)
- [CCI/DI+– COMBO indicator](#)
- [Dave Landry Pullbacks](#)
- [DMI Spread Index](#)
- [ekeko price chart](#)
- [Mndahoo ADX](#)
- [The Three Day Reversal](#)
- [Trend Analysis\\_Comentary](#)
- [Trending Ribbon](#)

#### **More information:**

[Updated on–line reference](#)

**PEAK****Basic price pattern detection****– peak**

(AFL 1.1)

**SYNTAX**     *peak*(ARRAY, change, n = 1)**RETURNS**     ARRAY

**FUNCTION**     Gives the value of ARRAY *n*-th peak(s) ago. This uses the Zig Zag function (see Zig Zag) to determine the peaks. *n* =1 would return the value of the most recent peak. *n* =2 would return the value of the 2nd most recent peak. **Caveat:** this function is based on Zig–Zag indicator and may look into the future.

**EXAMPLE**     peak(close,5,1)**SEE ALSO****Comments:**

<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2003-02-13 04:02:04	Zig/Peak/Trough functions work correctly for ARRAYS containing data greater than zero.
---	--

**References:**

The **PEAK** function is used in the following formulas in AFL on–line library:

- [Andrews Pitchfork](#)
- [Andrews PitchforkV3.3](#)
- [Double top detection](#)
- [ekeko price chart](#)
- [Fund Screener](#)
- [Gartley 222 Pattern Indicator](#)
- [MACD commentary](#)
- [Pivot Point with S/R Trendlines](#)
- [Schiff Lines](#)
- [Support Resistance levels](#)
- [Tom DeMark Trend Lines](#)

**More information:**

[Updated on–line reference](#)

**PEAKBARS****Basic price pattern detection****– bars since peak**

(AFL 1.1)

**SYNTAX**     *peakbars*(ARRAY, change, *n* = 1)**RETURNS**     ARRAY

**FUNCTION**     Gives the number of bars that have passed from the *n*-th peak. This uses the Zig Zag function (see Zig Zag) to determine the peaks. *n* =1 would return the number of bars that have passed since the most recent peak. *n* =2 would return the number of bars that have passed since the 2nd most recent peak **Caveat:** this function is based on Zig-Zag indicator and may look into the future.

**EXAMPLE**     *peakbars*(close,5,1)

**SEE ALSO****References:**

The **PEAKBARS** function is used in the following formulas in AFL on-line library:

- [DMI Spread Index](#)
- [Gartley 222 Pattern Indicator](#)
- [Head & Shoulders Pattern](#)
- [LunarPhase](#)
- [Pattern Recognition Exploration](#)
- [Pivot Point with S/R Trendlines](#)
- [QP2 Float Analysis](#)
- [RSI Trendlines and Wedges](#)
- [Stochastics Trendlines](#)
- [The Fibonacci behavior](#)
- [Tom DeMark Trend Lines](#)

**More information:**

[Updated on-line reference](#)

**PERCENTILE****Statistical functions****– calculate percentile**

(AFL 2.5)

**SYNTAX**     *Percentile( array, period, rank )***RETURNS**     ARRAY**FUNCTION**     The **Percentile** function gives *rank* percentile value of the array over last *period* bars.

rank is 0..100 – defines percentile rank in the array

Performance note: the implementation of percentile function involves sorting that is relatively slow process even though that quicksort algorithm is used.

**EXAMPLE**     `// Example 1:  
// show bars when 'current' Day Volume ranks within  
// TOP 30% of volumes of last 100 bars (is above 70th Percentile)  
Filter = Volume > Percentile( Volume, 100, 70 );  
  
// Example 2:  
// show bars when 'current' Day Volume ranks within  
// BOTTOM 30% of volumes of last 100 bars (is below 30th percentile)  
Filter = Volume < Percentile( Volume, 100, 30 );`

**SEE ALSO**     [Median\(\)](#) function**References:**

The **Percentile** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**PLOT****Exploration / Indicators****– plot indicator graph**

(AFL 1.8)

**SYNTAX** *Plot( array, name, color/barcolor, style = styleLine, minvalue = {empty}, maxvalue = {empty}, XShift = 0 )*

**RETURNS** NUMBER

**FUNCTION** Plots the graph using **array** data. Parameters:

- **array** – data array to be plotted
- **name** – defines graph name used for displaying values in a title bar.
- **color** – defines plot color that could be static (if third argument is a number) or dynamic (when third argument is an array). Color indexes are related to the current palette (see Preferences/Color)
- **style** is a combination of one or more of following values:

styleLine = 1 – normal (line) chart (default)

styleHistogram = 2 – histogram chart

styleThick = 4 – fat (thick)

styleDots = 8 – include dots

styleNoLine = 16 – no line

styleDashed = 32 – dashed line style

styleCandle = 64 – candlestick chart

styleBar = 128 – traditional bar chart

styleNoDraw = 256 – no draw (perform axis scaling only)

styleStaircase = 512 – staircase (square) chart

styleSwingDots = 1024 – middle dots for staircase chart

styleNoRescale = 2048 – no rescale

styleNoLabel = 4096 – no value label

stylePointAndFigure = 8192 – point and figure

(new in 4.20):

styleArea = 16384 – area chart (extra wide histogram)

styleOwnScale = 32768 – plot is using independent scaling

styleLeftAxisScale = 65536 – plot is using left axis scale (independent from right axis)

styleNoTitle – do not display values of this plot in the chart title

styleCloud – cloud style (area between high and low arrays) – to be used with PlotOHLC function

styleClipMinMax – clip (do not paint) area between min and max levels – note this style is incompatible with printers and WMF (metafiles).

- **minvalue** and **maxvalue** – (used by styleOwnScale plots ONLY) define plot minimum and maximum values (lower and upper boundary for Y axis)
- **XShift** – allows to visually shift the chart past the last bar.

**EXAMPLE**    *// Example 20-bar Moving average shifted 10 bars into the future  
past the last bar:  
Plot(Close, "Close", colorBlack, styleCandle);  
Plot(MA(Close, 20), "Shifted MA", colorRed, styleLine, Null, Null, 10  
);  
// Note that shift occurs during plotting AND does NOT affect source*

array

**SEE ALSO** [PLOTFOREIGN\(\)](#) function , [PLOTGRID\(\)](#) function , [PlotText\(\)](#) function , [PLOTSHAPES\(\)](#) function , [PLOTOHLC\(\)](#) function

#### References:

The **PLOT** function is used in the following formulas in AFL on–line library:

- [% B of Bollinger Bands With Adaptive Zones](#)
- ['DE\\*H37href='http://www.amibroker.com/library/detail.php?id=325target='\\_blank'>'R' Channel](#)
- [10–20 Indicator](#)
- [3 Price Break](#)
- [30 Week Hi Indicator – Display](#)
- [52 Week New High–New Low Index](#)
- [abosliman2005m](#)
- [AccuTrack](#)
- [Adaptave Zones O/B &O/S Oscillator](#)
- [Adaptive Laguerre Filter, from John Ehlers](#)
- [Adaptive Price Channel](#)
- [ADX Indicator – Colored](#)
- [ADXR](#)
- [AFL Example](#)
- [AFL Example – Enhanced](#)
- [AFL Timing functions](#)
- [Andrews Pitchfork](#)
- [Andrews PitchforkV3.3](#)
- [Another Flb Level](#)
- [Application of Ehler filter](#)
- [AR\\_Prediction.afl](#)
- [ATR Trading System](#)
- [Automatic Trend–line](#)
- [Balance of Power](#)
- [balance of power](#)
- [BMTRIX Intermediate Term Market Trend Indicator](#)
- [Bollinger – Keltner Bands](#)
- [Bollinger Band Width](#)
- [Bollinger Fibonacci Bands](#)
- [Bull/Bear Volume](#)
- [Bullish Percent Index 2 files combined](#)
- [Bullish Percent Index 2004](#)
- [Candle Identification](#)
- [Candle Stick Analysis](#)
- [Candle Stick Demo](#)
- [CCI Woodies Style](#)
- [CCI\(20\) Divergence Indicator](#)
- [Chande Momentum Oscillator](#)
- [Chande's Trend Score](#)
- [Chandelier Exit](#)
- [Chandelier Exit or Advanced Trailing Stop](#)
- [Chandelier Plugin](#)
- [Color Display.afl](#)



- Constant Trendline Plot
- Cycle Highlighter
- Cycle Highlighter (auto best-fit)
- Dahl Oscillator modified
- Darvas Box
- Dave Landry PullBack Scan
- Days to Third Friday
- Distance Coefficient Ehlers Filter
- Divergence indicator
- Donchian Channel
- DT Oscillator
- Dynamic Momentum Index
- Dynamic Momentum Index
- Ed Seykota's TSP: EMA Crossover System
- Ed Seykota's TSP: Support and Resistance
- Ehlers Center of Gravity Oscillator
- Ehlers CyberCycle
- Ehlers Dominant Cycle Period
- Ehlers Fisher Transform
- Ehlers Instantaneous Trend
- Ehlers Laguerre RSI
- ekeko price chart
- EKEKO SAR-MF
- Elder Impulse Indicator
- Elder Impulse Indicator V2
- Elder Ray – Bull Bear
- Elder safe Zone Long + short
- Elder Triple Screen Trading System.
- Elder's Market Thermometer
- Elder's SafeZone Stop
- Ema bands
- EMA Crossover
- EMA Crossover Price
- Fib CMO
- Fibonacci Moving averages
- fishnet
- Force index
- Future MA Projection
- Futures – Dollar Move Indicator
- Futures – Dollar Move Today Indicator
- Gartley 222 Pattern Indicator
- Gordon Rose
- Historical Volatility Scan – 6/100
- Historical Volatility Scan – 50 Day
- Hull Moving Average
- Hurst "Like" DE
- IFT of RSI – Multiple TimeFrames
- Index of 30 Wk Highs Vs Lows
- Indicator Explorer (ZigZag)
- INTRADAY HEIKIN ASHI new
- Lagging MA-Xover
- Linear Regression Line w/ Std Deviation Channels

- Log Time Scale
- Luna Phase
- LunarPhase
- MACD and histogram divergence detection
- MACD Histogram – Change in Direction
- MACD indicator display
- MAM
- Market Profile & Market Volume Profile
- Mndahoo ADX
- Modified Darvas Box
- Moving Averages NoX
- Moving Trend Bands (MTB)
- MultiCycle 1.0
- Multiple sinus noised
- nifty
- nikhil
- Nonlinear Ehlers Filter
- Noor\_Doodie
- nth ( 1 – 8 ) Order Polynomial Fit
- Option Calls, Puts and days till third friday.
- PFChart – High/Low prices Sept2003
- PFchart with range box sizes
- Parabolic SAR in native AFL (v.4.31.1 required)
- ParabXO
- Performance Overview
- PF Chart – Close – April 2004
- Pivot Point with S/R Trendlines
- Pivots for Intraday Forex Charts
- Polarized Fractal Efficiency
- Position Sizing and Risk Price Graph
- Position Sizing and Risk Price Graph – 2
- prakash
- Price with Woodies Pivots
- Projection Oscillator
- R–Squared
- Random Walk
- Ranking and sorting stocks
- Ranking Ticker WatchList
- Rea Time Daily Price Levels
- Relative Strength Multichart of up to 10 tickers
- Renko Chart
- Reverse EMA function
- RSI of volume weighted moving average
- RSI Trendlines and Wedges
- RSIS
- RUTVOL timing signal with BB Scoring routine
- SAR–ForNextBarStop
- Schiff Lines
- Second–order Infinite impulse response filter
- Shares To Buy Price Graph
- Simple Momentum
- Standard Error Bands (Native AFL)

- Steve Woods' Cum. Vol. Float + Cum. Vol. Channels
- Steve Woods' Cumulative Vol. Percentage Indicator
- Stochastic of Weekly Price Array
- Support Resistance levels
- T3 Function
- TD sequential
- The D\_oscillator
- The Fibonacci behavior
- The Saturation Indicator D\_sat
- The Stochastic CCI
- Time Frame Weekly Bars
- Time segment value
- Tom DeMark Trend Lines
- tomy\_frenchy
- Trading ATR 10-1
- Trend Detection
- Trend Trigger Factor
- Trending Ribbon
- Triangular Moving Average
- Triangular Moving Average new
- Trigonometric Fit – TrigFit with AR for cos / sin
- Twiggs Money Flow
- TWS auto-export Executions-file parser
- Ultimate plus
- Using From and To dates from Auto Analysis in Code
- VAMA
- Vertical Horizontal Filter (VHF)
- Visualization of stoploses and profit in chart
- Volatility System
- Volume – compared with Moving Avg (100%)
- William's % R
- Williams %R Exploration
- Woodies CCI
- ZeroLag MACD(p,q,r)
- Zig Zag
- Zig Zag Indicator with Valid Entry and Exit Points
- ZigZag filter rewritten from scratch in AFL
- \_Volume

**More information:**

Updated on-line reference

**PLOTFOREIGN****– plot foreign security data**Referencing other symbol data  
(AFL 2.2)

**SYNTAX**      *PlotForeign( tickersymbol, name, color/barcolor, style = styleCandle / styleOwnScale, minvalue = {empty}, maxvalue = {empty}, XShift = 0 )*

**RETURNS**      NUMBER

**FUNCTION**      Plots the foreign–symbol price chart (symbol is defined by *tickersymbol* parameter). Second argument *name* defines graph name used for displaying values in a title bar. Graph color could be static (if third argument is a number) or dynamic (when third argument is an array). Color indexes are related to the current palette (see Preferences/Color)  
*style* defines chart plot style (see Plot() function for possible values)  
*minvalue* and *maxvalue* – (used by styleOwnScale plots ONLY) define plot minimum and maximum values (lower and upper boundary for Y axis)  
*XShift* – allows to visually shift the chart into future (blank) bars.

**EXAMPLE**      PlotForeign( "^DJI", "Dow Jones", colorRed );

**SEE ALSO****References:**

The **PLOTFOREIGN** function is used in the following formulas in AFL on–line library:

- [Dave Landry PullBack Scan](#)
- [Elder Triple Screen Trading System](#).

**More information:**

[Updated on–line reference](#)

**PLOTGRID****Exploration / Indicators****– Plot horizontal grid line**

(AFL 2.3)

**SYNTAX**     *PlotGrid( level, color = colorDefault )***RETURNS**    NOTHING**FUNCTION**   Plots horizontal grid line using built-in dotted style at given level and color.**EXAMPLE**    `PlotGrid( 25, colorRed );`**SEE ALSO**    [PLOT\(\)](#) function , [PLOTFOREIGN\(\)](#) function , [PLOTOHLC\(\)](#) function**Comments:**

<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2003-04-18 07:12:14	<p>Instead of number you can also use expression but it must be NUMERIC expression, not ARRAY.</p> <p>Use LastValue to convert:</p> <p>your_expression = ...  PlotGrid( LastValue( your_expression ) );</p>
--	---

**References:**

The **PLOTGRID** function is used in the following formulas in AFL on-line library:

- [CCI Woodies Style](#)
- [Dynamic Momentum Index](#)
- [Dynamic Momentum Index](#)
- [Ehlers Center of Gravity Oscillator](#)
- [Ehlers Fisher Transform](#)
- [Ehlers Laguerre RSI](#)
- [IFT of RSI – Multiple TimeFrames](#)
- [MultiCycle 1.0](#)
- [PFchart with range box sizes](#)

**More information:**

[Updated on-line reference](#)

**PLOTOHLC****Exploration / Indicators****– plot custom OHLC chart**

(AFL 2.2)

**SYNTAX**     *PlotOHLC( open, high, low, close, name, color/barcolor, style = styleCandle / styleOwnScale, minvalue = {empty}, maxvalue = {empty}, XShift = 0 )*

**RETURNS**     NUMBER

**FUNCTION**     Plots the price chart using custom *open, high, low, close* arrays supplied as parameters. Fifth argument *name* defines graph name used for displaying values in a title bar. Graph color could be static (if sixth argument is a number) or dynamic (when sixth argument is an array). Color indexes are related to the current palette (see Preferences/Color)  
*style* defines chart plot style (see Plot() function for possible values)  
*minvalue* and *maxvalue* – (used by styleOwnScale plots ONLY) define plot minimum and maximum values (lower and upper boundary for Y axis)  
*XShift* – allows to visually shift the chart into future (blank) bars.

**EXAMPLE**     PlotOHLC( 1.1\*Open, 1.1\* High, 1.1\* Low, 1.1\* Close, "Price chart shifted 10% up", colorRed, styleCandle );

**SEE ALSO**     [PLOT\(\)](#) function , [PLOTFOREIGN\(\)](#) function

**References:**

The **PLOTOHLC** function is used in the following formulas in AFL on–line library:

- [Bollinger Fibonacci Bands](#)
- [Constant Trendline Plot](#)
- [DPO with shading](#)
- [Elder Triple Screen Trading System.](#)
- [Gordon Rose](#)
- [Hurst "Like" DE](#)
- [Indicator Explorer \(ZigZag\)](#)
- [INTRADAY HEIKIN ASHI new](#)
- [Log Time Scale](#)
- [MACD Histogram – Change in Direction](#)
- [Monthly bar chart](#)
- [N–period candlesticks \(time compression\)](#)
- [nth \( 1 – 8 \) Order Polynomial Fit](#)
- [Pivot Finder](#)
- [Pivots for Intraday Forex Charts](#)
- [Price with Woodies Pivots](#)
- [Rea Time Daily Price Levels](#)
- [Renko Chart](#)
- [Three Line Break – TLB](#)
- [Weekly chart](#)

**More information:**

[Updated on–line reference](#)

**PLOTSHAPES****Exploration / Indicators****– plots arrows and other shapes**

(AFL 2.3)

**SYNTAX** *PlotShapes( shape, color, layer = 0, yposition = graph0, offset = -12 );***RETURNS** NOTHING**FUNCTION** Plots arrows and other shapes on any chart pane.

Parameters:

- shape defines type of the symbol. when shape is zero nothing is plotted values other than zero cause plotting various pre-defined shapes. Odd values plot shape BELOW indicator, even values plot shape ABOVE indicator.
- color defines color of shape
- layer defines layer number on which shapes are plotted
- yposition defines Y-position where shapes are plotted (by default they are plotted 'around' graph0 (first indicator) line)
- offset – (or distance) parameter (by default -12 ), Offset is expressed in SCREEN pixels. Negative offsets shift symbols down, positive offsets shift symbol up. To place the shape exactly at yposition, specify 0 as offset

Constants for shapes:

shapeNone, shapeUpArrow, shapeDownArrow, shapeHollowUpArrow,  
 shapeHollowDownArrow, shapeSmallUpTriangle, shapeSmallDownTriangle,  
 shapeHollowSmallUpTriangle, shapeHollowSmallDownTriangle, shapeUpTriangle,  
 shapeDownTriangle, shapeHollowUpTriangle, shapeHollowDownTriangle,  
 shapeSmallSquare, shapeHollowSmallSquare, shapeSquare, shapeHollowSquare,  
 shapeSmallCircle, shapeHollowSmallCircle, shapeCircle, shapeHollowCircle, shapeStar,  
 shapeHollowStar, shapeDigit0, shapeDigit1, shapeDigit2, shapeDigit3, shapeDigit4,  
 shapeDigit5, shapeDigit6, shapeDigit7, shapeDigit8, shapeDigit9, shapePositionAbove

**EXAMPLE** Example 1:

```
PlotShapes( IIF( buy, shapeDigit9 + shapePositionAbove, shapeNone ),
  colorGreen );
```

Example 2:

```
Graph0=MACD();
Graph1=Signal();
Buy=Cross(Graph0, Graph1);
Sell=Cross(Graph1, Graph0);
PlotShapes( ( Buy OR Sell ) * ( 1 + Cum( Buy OR Sell ) % 52 ), IIF(
  Buy, colorGreen, colorRed ), 5 );
GraphXSpace = 5;
```

**SEE ALSO** [PLOT\(\)](#) function

**Comments:**

<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2003-07-01 09:31:04	You can position your arrows relative to High/Low too. See the code below for the example:  Buy=Cross(MACD(), Signal());  Sell=Cross(Signal(), MACD());  shape = Buy * shapeUpArrow + Sell * shapeDownArrow;  Plot( Close, "Price", colorBlack, styleCandle );  PlotShapes( shape, If( Buy, colorGreen, colorRed ), 0, If( Buy, Low, High ) );  GraphXSpace = 5;
<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2006-06-07 17:14:40	ShapePositionAbove must NOT be used together with shapes that have positions already included (all those which have "Down" or "Up" in the name). All "Down" shapes are positioned ABOVE already and "Up" shapes are positioned BELOW already, so it makes no sense to add those two.  So you may only use ShapePosition above to the following shapes: shapeSmallSquare, shapeHollowSmallSquare, shapeSquare, shapeHollowSquare, shapeSmallCircle, shapeHollowSmallCircle, shapeCircle, shapeHollowCircle, shapeStar, shapeHollowStar, shapeDigit0, shapeDigit1, shapeDigit2, shapeDigit3, shapeDigit4, shapeDigit5, shapeDigit6, shapeDigit7, shapeDigit8, shapeDigit9

**References:**

The **PLOTSHAPES** function is used in the following formulas in AFL on-line library:

- [DE\\*H37](http://www.amibroker.com/library/detail.php?id=718S)<http://www.amibroker.com/library/detail.php?id=718S> target='\_blank'>abosliman2005m
- [AFL Example](#)
- [AFL Example – Enhanced](#)
- [An n bar Reversal Indicator](#)
- [ATR Trading System](#)
- [Candle Stick Analysis](#)
- [Chandelier Exit or Advanced Trailing Stop](#)
- [Dynamic Momentum Index](#)
- [Dynamic Momentum Index](#)
- [ekeko price chart](#)
- [EKEKO SAR–MF](#)
- [Elder Impulse Indicator](#)
- [Elder Impulse Indicator V2](#)
- [Elder safe Zone Long + short](#)
- [Expiry Thursday for Indian markets](#)
- [Gordon Rose](#)
- [Hull Moving Average](#)
- [Hurst "Like" DE](#)



- [IFT of RSI – Multiple TimeFrames](#)
- [Indicator Explorer \(ZigZag\)](#)
- [Lagging MA–Xover](#)
- [LunarPhase](#)
- [MACD and histogram divergence detection](#)
- [Moving Averages NoX](#)
- [Pivot Finder](#)
- [RUTVOL timing signal with BB Scoring routine](#)
- [TD sequential](#)
- [Trading ATR 10–1](#)
- [Trend Detection](#)
- [Triangular Moving Average new](#)
- [TWS auto–export Executions–file parser](#)
- [Using From and To dates from Auto Analysis in Code](#)
- [Visualization of stoploses and profit in chart](#)
- [Volatility System](#)
- [Zig Zag Indicator with Valid Entry and Exit Points](#)

**More information:**

[Updated on–line reference](#)

**PLOTTEXT****– write text on the chart****Indicators**  
(AFL 2.80)**SYNTAX** *PlotText( "text", x, y, color, bgcolor = colorDefault )***RETURNS** NOTHING**FUNCTION** This function writes text in specified co-ordinates.

where:

- **x** – is x-coordinate in bars (like in LineArray)
- **y** – is y-coordinate in dollars
- **color** is text color
- **bgcolor** is background color

If bgcolor is NOT specified (or equal to colorDefault) text is written with TRANSPARENT background, any other value causes solid background with specified background color.

**EXAMPLE**

```

Plot(C,"Price", colorBlack, styleLine );
Plot(MA(C,20),"MA20", colorRed );

Buy=Cross( C, MA(C,20) );
Sell= Cross( MA( C, 20 ), C );

dist = 1.5*ATR(10);

for( i = 0; i < BarCount; i++ )
{
    if( Buy[i] ) PlotText( "Buy\n@" + C[ i ], i, L[ i ]-dist[i],
        colorGreen );
    if( Sell[i] ) PlotText( "Sell\n@" + C[ i ], i, H[ i ]+dist[i],
        colorRed, colorYellow );
}

PlotShapes( Buy * shapeUpArrow + Sell * shapeDownArrow, IIf( Buy,
    colorGreen, colorRed ) );

```

**SEE ALSO** [PLOT\(\)](#) function**References:**

The **PlotText** function is used in the following formulas in AFL on-line library:

- [SUPER PIVOT POINTS](#)
- [Visualization of stoploses and profit in chart](#)

**More information:**

[Updated on-line reference](#)

**PLOTVAPOVERLAY****Exploration / Indicators****– plot Volume–At–Price overlay chart**

(AFL 2.4)

**SYNTAX** *PlotVAPOverlay( lines = 300, width = 5, color = colorGreen, vapstyle = 0 );***RETURNS** NOTHING**FUNCTION** Plots Volume–At–Price (VAP) overlay chart. Please note that there must be at least one regular Plot function in your formula for this to work, and there can be only one PlotVAPOverlay in one indicator

- vapstyle = 0 – left side, area fill, on top of all plots
- vapstyle = 1 – right side, area fill, on top of all plots
- vapstyle = 2 – left side, lines only, on top of all plots
- vapstyle = 3 – right side, lines only, on top of all plots
- vapstyle = 4 – left side, area fill, behind all plots
- vapstyle = 5 – right side, area fill, behind all plots
- vapstyle = 6 – left side, lines only, behind all plots
- vapstyle = 7 – right side, lines only, behind all plots

**EXAMPLE** `Plot( Close, "Price", colorWhite, styleCandle );`  
`PlotVAPOverlay( Param( "lines", 300, 10, 1000, 1 ),`  
`Param( "width", 10, 1, 99, 1 ), ParamColor( "color", colorDarkBlue ),`  
`Param( "style", 0, 0, 7, 1 ) );`

**SEE ALSO** [PLOT\(\)](#) function**References:**The **PLOTVAPOVERLAY** function is used in the following formulas in AFL on–line library:**More information:**[Updated on–line reference](#)

## POPUPWINDOW

### – display pop-up window

Miscellaneous functions  
(AFL 3.0)

**SYNTAX**     *PopupWindow( bodytext, captiontext, timeout = 5, left = -1, top = -1 );*

**RETURNS**    NOTHING

**FUNCTION**    The function creates and displays pop-up window with specified bodytext, captiontext.

Parameters:

- bodytext – the string containing the text of the window body
- caption – the string containing the text of window caption
- timeout – auto-close time in seconds (default 5 seconds)
- left – top-left corner X co-ordinate (default = -1 –means auto-center)
- top – top-left corner Y co-ordinate (default = -1 – means auto-center)

**EXAMPLE**

```
if( ParamTrigger( "Display Popup Window", "Press here" ) )
{
    PopupWindow( "Current time is: " + Now(), "Alert", 2,
640*mtRandom(), 480*mtRandom() );
}
```

**SEE ALSO**     ParamColor() function , ParamDate() function , ParamField() function , ParamList() function , PARAMSTR() function , ParamStyle() function , ParamTime() function , ParamToggle() function , ParamTrigger() function

#### References:

The **PopupWindow** function is used in the following formulas in AFL on-line library:

#### More information:

[Updated on-line reference](#)

**PREC****Math functions****– adjust number of decimal points of floating point number****SYNTAX**     *prec*(*ARRAY, precision* )**RETURNS**     ARRAY**FUNCTION**     Truncates ARRAY to *precision* decimal places.**EXAMPLE**     The formula "prec( 10.12981, 2 )" returns 10.120. The formula "prec( 10.12981, 4 )" returns 10.12980.**SEE ALSO****References:**

The **PREC** function is used in the following formulas in AFL on–line library:

- [CCI Woodies Style](#)
- [Relative Strength Multichart of up to 10 tickers](#)
- [Triangle exploration using PFChart](#)

**More information:**

[Updated on–line reference](#)

**PREFS****– retrieve preferences settings****Miscellaneous functions**  
(AFL 1.4)**SYNTAX**     *prefs( index )***RETURNS**    NUMBER, STRING**FUNCTION**    Retrieves preferences setting. Allowed *index* values are:

0: FatLineChart;  
 1: MarkQuotations;  
 2: ChartVolumeType;  
 3: ShortTimeMA;  
 4: STMARange;  
 5: MidTimeMA;  
 6: MTMARange;  
 7: BollingerBands;  
 8: Pref.BBFactor;  
 9: ROC;  
 10: RSI;  
 11, 12, 13: MACD;  
 14: StochSlow;  
 15, 16, 17: Ultimate;  
 18: VolumeType;  
 19–22: /\* reserved amiga only \*/  
 23: AutoArrange;  
 24: LogChartScale;  
 25: MaxChartQuot;  
 26: TRIX;  
 27: LongTimeMA;  
 28: LTMARange;  
 29: VolMARange;  
 30: RelativeStrengthBase (string);  
 31: LimitSave;  
 32: LimitSaveRange;  
 33: CCI;  
 34: CCIAvg;  
 35: Tooltips;  
 36: MFI;  
 37, 38: Chaikin;  
 39: DataPath (string)  
 40: DataTooltips;  
 41: LoadAllWhenSelect;  
 42: PartialLoad;  
 43: PartialLoadQty;  
 44, 45: TRIN;  
 46: STMAType;  
 47: MTMAType;  
 48: LTMAType;  
 49: ADX;  
 50, 51: ParabolicSAR;  
 52: EnableMainChartSAR;

53: DefaultPriceStyle;  
54: StockTreeMode;  
55: TickerListMode;

**EXAMPLE**    `macd( prefs( 11 ), prefs( 12 ), prefs( 13 ) );`

**SEE ALSO**

**References:**

The **PREFS** function is used in the following formulas in AFL on-line library:

- [Dinapoli Guru Commentary](#)
- [ekeko price chart](#)
- [hassan](#)
- [Moving Trend Bands \(MTB\)](#)

**More information:**

[Updated on-line reference](#)

**PRINTF****String manipulation****– Print formatted output to the output window.**

(AFL 2.5)

**SYNTAX**     *printf( formatstr, ... )***RETURNS**    NOTHING**FUNCTION**    The **printf** function formats and prints a series of characters and values to the output window, which can be either commentary or interpretation window.

If arguments follow the format string, the format string must contain specifications that determine the output format for the arguments.

printf and StrFormat behave identically except that printf writes output to the window, while StrFormat does not write anything to output window but returns resulting string instead.

Note 1: for numbers always use %f, %e or %g formatting, %d or %x will not work because there are no integers in AFL.

Note 2: as of now only numbers and arrays can now be printed. For arrays 'selected value' is printed

**EXAMPLE**     `for( i = 0; i < 10; i++ )`  
                   {  
                     printf( "Hello world, line %g\n", i );  
                   }

**SEE ALSO**     [StrFormat\(\)](#) function**References:**

The **printf** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [DateNum\\_DateStr](#)
- [ekeko price chart](#)
- [Multiple sinus noised](#)
- [nth \( 1 – 8 \) Order Polynomial Fit](#)
- [prakash](#)
- [tomy\\_frenchy](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [Ultimate plus](#)
- [WLBuildProcess](#)

**More information:**

[Updated on–line reference](#)



**PVI****Indicators****– positive volume index****SYNTAX**     *pvi()***RETURNS**     ARRAY**FUNCTION**     Calculates the Positive Volume Index.**EXAMPLE****SEE ALSO**     The *nvi()* function**References:**

The **PVI** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

## RANDOM

### – random number

**Statistical functions**  
(AFL 1.9)

**SYNTAX**     *random( seed = Null )*

**RETURNS**    ARRAY

**FUNCTION**    Returns an array of random values in 0..1 range (to get single random value use LastValue( Random() ) )  
Seed is defined it initializes the seed of random number generator this allows to produce repetitive series of pseudo-random series. If seed is not specified – random number generator continues generation. To reinitialize the generator, use 1 as the seed argument. Any other value for seed sets the generator to a random starting point.

**EXAMPLE**     Example 1:

```
Graph0 = Random(); // generates different sequence with each refresh
```

Example 2:

```
Graph0 = Random(1); // generates the same sequence with each refresh
```

**SEE ALSO**     [mtRandom\(\)](#) function

#### Comments:

<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2007-02-27 09:42:14	Internally Random() function uses Microsoft C runtime library rand() function scaled to cover 0..1 range: (1.0*rand()/RAND_MAX)  Now Microsoft's rand() function (used in all MS languages) is Linear Congruential Pseudo-Random Number Generator coded using 32 bit integer arithmetic as follows:  <pre>static long holdrand; int rand() {     holdrand = holdrand * 214013 + 2531011;     return ( holdrand &gt;&gt; 16 ) &amp; 0x7fff; }</pre>
--	---

#### References:

The **RANDOM** function is used in the following formulas in AFL on-line library:

- [Monte Carlo](#)
- [Multiple sinus noised](#)
- [Random Walk](#)
- [Randomize\(\)](#)

#### More information:

[Updated on-line reference](#)



**REF****Trading system toolbox****– reference past/future values of the array****SYNTAX**     *ref( ARRAY, period )***RETURNS**     ARRAY**FUNCTION**     References a previous or subsequent element in a ARRAY. A positive *period* references "n" periods in the future; a negative *period* references "n" periods ago.**EXAMPLE**     The formula "ref( CLOSE, -14 )" returns the closing price 14 periods ago. Thus, you could write the 14-day price rate-of-change (expressed in points) as "C – ref( C, -14 )." The formula "ref( C, 12 )" returns the closing price 12 periods ahead (this means looking up the future)**SEE ALSO****References:**

The **REF** function is used in the following formulas in AFL on-line library:

- [3 Price Break](#)
- [30 Week Hi Indicator – Calculate](#)
- [52 Week New High–New Low Index](#)
- [a](#)
- [AC+ acceleration](#)
- [Adaptave Zones O/B &O/S Oscillator](#)
- [Adaptive Price Channel](#)
- [ADX Indicator – Colored](#)
- [ADXbuy](#)
- [ADXR](#)
- [AFL Example](#)
- [AFL Example – Enhanced](#)
- [Against all odds](#)
- [AJDX system](#)
- [Alert Output As Quick Rewiev](#)
- [An n bar Reversal Indicator](#)
- [Analytic RSI formula](#)
- [Andrews Pitchfork](#)
- [Andrews PitchforkV3.3](#)
- [AO+ Momentum indicator](#)
- [Application of Ehler filter](#)
- [Aroon](#)
- [AR\\_Prediction.afl](#)
- [ATR Study](#)
- [Auto Analysis Closing Price Reversal](#)
- [Auto Analysis Hook Reversal](#)
- [Auto Analysis Island Reversal](#)
- [Auto Analysis Key Reversal](#)
- [Auto Analysis Open/Close Reversal](#)
- [Auto Analysis Pivot Point Reversal](#)
- [Auto Analysis Short-term Reversals Exploration](#)
- [Auto–Optimization Framework](#)

- Balance of Power
- balance of power
- Bollinger Band Gap
- Bollinger band normalization
- Bollinger Fibonacci Bands
- Bow tie
- Buff Volume Weighted Moving Averages
- Bull/Bear Volume
- Bullish Percent Index 2 files combined
- Bullish Percent Index 2004
- CAMSLIM Cup and Handle Pattern AFL
- Candle Identification
- Candle Pattern Function
- Candle Stick Analysis
- CandleStick Comentary---Help needed
- Candlestick Commentary
- Candlestick Commentary Modified
- Candlestick Commentary-modified
- Candlestick Volume Bars with Moving Average
- CandleStochastics
- CCI Woodies Style
- CCI(20) Divergence Indicator
- CCT Coppock Curve
- CCT Kaleidoscope
- Chande Momentum Oscillator
- Chande's Trend Score
- Cole
- Color Coded Short Term Reversal Signals
- Commodity Channel Index
- Compare Sectors against Tickers
- crBeta
- Cycle Highlighter
- Cycle Highlighter (auto best-fit)
- Dahl Oscillator modified
- danningham penetration
- Dave Landry PullBack Scan
- Dave Landry Pullbacks
- Demand Index
- DeMarker
- Dinapoli Guru Commentary
- Distance Coefficient Ehlers Filter
- Divergence indicator
- Divergences
- DMI Spread Index
- Donchian Channel
- Double top detection
- DPO with shading
- Dynamic Momentum Index
- Dynamic Momentum Index
- Ed Seykota's TSP: EMA Crossover System
- Ehler's filters and indicators
- Ehlers Center of Gravity Oscillator

- Ehlers CyberCycle
- Ehlers Dominant Cycle Period
- Ehlers Fisher Transform
- Ehlers Instantaneous Trend
- EKEKO SAR–MF
- Elder Impulse Indicator
- Elder Impulse Indicator V2
- Elder safe Zone Long + short
- Elder Triple Screen Trading System.
- Elder's Market Thermometer
- Elder's SafeZone Stop
- ElderSafeZoneStopLong
- ElderSafeZoneStopShort
- Ema bands
- EMA Crossover
- End Of Year Trading
- Evaluating Candle Patterns in a trading system
- FastStochK FullStochK–D
- Fib CMO
- Follow the Leader
- Force index
- Fund Screener
- Future MA Projection
- Futures – Dollar Move Today Indicator
- Gann Five Day pullback
- Gann HiLo Indicator and System
- Gann Swing Chart
- garythompson
- garythompson
- Gordon Rose
- hassan
- Hilbert Study
- Historical Volatility Scan – 6/100
- Historical Volatility Scan – 50 Day
- Hook Reversals
- Hurst "Like" DE
- Hurst Constant
- Ichimoku Chart
- Ichimoku charts
- IFT of RSI – Multiple TimeFrames
- Index of 30 Wk Highs Vs Lows
- Indicator Explorer (ZigZag)
- INTRADAY HEIKIN ASHI new
- IntraDay Open Marker
- Kagi Chart
- Lagging MA–Xover
- Larry William's Volatility Channels
- Linear Regression Line w/ Std Deviation Channels
- MACD and histogram divergence detection
- MACD commentary
- MACD indicator display
- Main price chart with Rainbow & SAR

- Market Direction
- Market Profile & Market Volume Profile
- Modified Darvas Box
- Modified Momentum Finder DDT–NB
- Momentum
- Momentum
- Monthly bar chart
- Monthly Coppock Guide
- Moving Average "Crash" Test
- Moving Averages NoX
- MultiCycle 1.0
- N–period candlesticks (time compression)
- Negative ROC Exporation
- Nonlinear Ehlers Filter
- Noor\_Doodie
- NR4 Historical Volatility System
- PFChart – High/Low prices Sept2003
- PFchart with range box sizes
- Pattern – Rectangle Base Breakout on High Vol
- Performance Check
- Performance Overview
- Peterson
- PF Chart – Close – April 2004
- Pivot Point with S/R Trendlines
- Pivots for Intraday Forex Charts
- Plot Monthly, Weekly and Daily Moving average
- Polarized Fractal Efficiency
- Position Sizing and Risk Price Graph
- Position Sizing and Risk Price Graph – 2
- Positive ROC Exploration
- Price Persistency
- Price with Woodies Pivots
- Price–Volume Rank
- Probability Calculator
- Projection Oscillator
- Pullback System No. 1
- Rainbow Charts
- Rainbow Oscillator
- Range Expansion Index
- Raw ADX
- Rea Time Daily Price Levels
- Regression Analysis Line
- Relative Strength Multichart of up to 10 tickers
- Relative Vigour Index
- Renko Chart
- Reverse EMA function
- ROC of MACD Weekly
- RSI of Weekly Price Array
- RSIS
- RUTVOL timing signal with BB Scoring routine
- Sainath Sidgiddi
- Schiff Lines

- Sector Tracking
- SectorRSI
- Simple Momentum
- Steve Woods' Cum. Vol. Float + Cum. Vol. Channels
- Stochastic Divergence, negative
- Stochastic Divergence, positive
- Stochastic Divergences, PDI, NDI
- Stochastic Fast%K and Full
- Stochastic of Weekly Price Array
- Stops Implementation in AFS
- Strength and Weakness
- SUPER PIVOT POINTS
- TAZ Trading Method Exploration
- TD sequential
- The Mean RSI
- The Mean RSI (variations)
- The Relative Slope
- The Relative Slope Pivots
- The Saturation Indicator D\_sat
- The Three Day Reversal
- Three Line Break – TLB
- Time Frame Weekly Bars
- Time segment value
- Tom DeMark Trend Lines
- tomy\_frenchy
- Trading ATR 10–1
- Trend Analysis\_Comentary
- Trend Continuation Factor
- Trend Detection
- Trend Trigger Factor
- Triangle exploration using PFChart
- Triangle search
- Triangular Moving Average
- Triangular Moving Average new
- Trigonometric Fit – TrigFit with AR for cos / sin
- TRIX
- Tushar Chande's Projected Range
- Twiggs Money Flow
- Varexlist
- Vertical Horizontal Filter
- Vertical Horizontal Filter (VHF)
- Vic Huebner
- Volatility Quality Index
- Volatility System
- Volume Oscillator
- Weekly chart
- Weekly Trend in Daily Graph
- Weinberg's The Range Indicator
- William's % R
- Williams %R Exploration
- Williams Alligator system
- Woodies CCI



- [Zig Explorer](#)
- [Zig Zag Indicator with Valid Entry and Exit Points](#)

**More information:**

[Updated on-line reference](#)

**RELSTRENGTH****– comparative relative strength**Referencing other symbol data  
(AFL 1.3)**SYNTAX**      *relstrength( "tickername", fixup = 1)***RETURNS**    ARRAY**FUNCTION**    Calculates relative strength of currently selected security compared to "tickername" security. When you give an empty string as argument, a standard relative strength base security taken from Stock→Categories will be used.

The last parameter – fixup – with the default value of 1 – causes filling the holes in the data with previous values (behaviour introduced in 3.90.3), if fixup is 0 – the holes are not fixed (the old, pre-3.90.3 behaviour)

Note: you can still use Foreign/RelStrength in the old way:

Foreign( "ticker", "field" ), RelStrength( "ticker" ) – then the holes will be fixed.

**EXAMPLE**      relstrength( "^DJI" )**SEE ALSO****Comments:**

jayson	Interpretation
2003-06-23 09:20:02	<p>Comparative Relative Strength compares a security's price change with that of a "base" security. When the Comparative Relative Strength indicator is moving up, it shows that the security is performing better than the base security. When the indicator is moving sideways, it shows that both securities are performing the same (i.e., rising and falling by the same percentages). When the indicator is moving down, it shows that the security is performing worse than the base security (i.e., not rising as fast or falling faster).</p> <p>Comparative Relative Strength is often used to compare a security's performance with a market index. It is also useful in developing spreads (i.e., buy the best performer and short the weaker issue).</p>

**References:**The **RELSTRENGTH** function is used in the following formulas in AFL on-line library:

- [Relative Strength](#)
- [Relative strength comparison with moving average](#)

**More information:**[Updated on-line reference](#)

**REQUESTTIMEDREFRESH****Indicators**  
(AFL 2.90)**– forces periodical refresh of indicator pane****SYNTAX**     *RequestTimedRefresh( interval, onlyvisible = True )***RETURNS**    NOTHING**FUNCTION**    The function causes given indicator window to refresh automatically every seconds regardless of data source used or connection state.

**interval** parameter defines timeout in seconds between refreshes. AmiBroker attempts to align refreshes to second boundary so if you call it `RequestTimedRefresh( 5 )` you should get refreshes at 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 and 55 second of the minute. Due to the way how regular (low overhead) timers are implemented in Windows they have accuracy of +/-55ms provided that CPU is not very busy. Don't expect to get first line of your code to execute exactly at .000 milliseconds. This varies depending on machine load, number of quotes, system time slice and tens of other factors. Usually (on my testing machines) the first line of the code executes anywhere in the first 100 ms of the second, provided that other processes do not interfere. Windows is not real-time operating system and it does not guarantee any fixed execution/reaction times.

**onlyvisible** parameter set to True (default value) means that refreshes are triggered only for visible and not minimised windows. This applies also to main AmiBroker window – when it is minimised charts are NOT refreshed by default. To force refreshes when window is minimised you need to set this parameter to False. Note that this visibility applies to mostly to 'minimised' state or the situation when you move chart outside the boundary of physical screen so it is not visible to an eye but still open. It does not apply to chart windows that are on placed on inactive sheets, as they do not really exist until they are shown (this way AmiBroker conserves memory and CPU) and as non-existing, can not be refreshed.

Hint: to detect whenever given refresh comes from timer or user action you can use `Status("redrawaction")` function. It returns 0 for regular refresh (user action) and 1 for timer-refresh

**EXAMPLE**     `RequestTimedRefresh( 5 );`  
                   *// automatically refresh this particular chart every 5 seconds*

**SEE ALSO**     `STATUS()` function**References:**

The **RequestTimedRefresh** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)

**RESTOREPRICEARRAYS****– restore price arrays to original symbol****Referencing other symbol data**  
(AFL 2.5)**SYNTAX**     *RestorePriceArrays( tradeprices = False )***RETURNS**    NOTHING**FUNCTION**    The **RestorePriceArrays** restores original price and volume arrays after the call to **SetForeign**.*tradeprices* parameter has to match the one used in SetForeign() function.

When *tradeprices* argument is set to TRUE, then not only OHLC, V, OI, Avg arrays are restored, but BuyPrice, SellPrice, ShortPrice, CoverPrice, PointValue, TickSize, RoundLotSize, MarginDeposit variables too.

**EXAMPLE**     *// Example 1: Plot the indicator using foreign security data*  
                   *SetForeign( "MSFT" );*  
                   *Plot( Ultimate(), "Ultimate from MSFT", colorRed );*  
                   *RestorePriceArrays();*

*// Example 2: Use SetForeign with Equity function*  
                   *SetForeign( "MSFT", True, True );*  
                   *Buy = Cross( MACD(), Signal() );*  
                   *Sell = Cross( Signal(), MACD() );*  
                   *e = Equity(); // backtest on MSFT*  
                   *RestorePriceArrays( True ); //*

**SEE ALSO**    [SetForeign\(\)](#) function**Comments:**

<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2004-07-10 07:03:50	TimeFrameRestore and RestorePriceArrays is essentially the same function. So please note that calling RestorePriceArrays also resets the time interval set by eventual previous call to TimeFrameSet
--	--

**References:**

The **RestorePriceArrays** function is used in the following formulas in AFL on-line library:

- [Ranking and sorting stocks](#)
- [Ranking Ticker WatchList](#)

**More information:**[Updated on-line reference](#)

**RMI**  
**– Relative Momentum Index****Indicators**  
(AFL 2.1)**SYNTAX**     *rmi( periods = 20, momentum = 5 )***RETURNS**     ARRAY**FUNCTION**     Calculates Altman's Relative Momentum Index (SCFeb 1993)**EXAMPLE**     rmi( 20, 5 )**SEE ALSO****References:**

The **RMI** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**ROC****Indicators****– percentage rate of change****SYNTAX**     *roc( ARRAY, periods = 12, absmode = False )***RETURNS**     ARRAY

**FUNCTION**     Calculates the *periods* rate-of-change of ARRAY expressed as percentage.  
                      if absmode = False the value returned is array – ref( array, –periods )/ref( array, –periods )  
                      if absmode = True the value returned is array – ref( array, –periods )/abs( ref( array, –periods ) )

**EXAMPLE**     The formula "roc( CLOSE, 14 )" returns the 14–period percent rate-of-change of the closing prices.

**SEE ALSO****References:**

The **ROC** function is used in the following formulas in AFL on–line library:

- [AccuTrack](#)
- [Adaptave Zones O/B &O/S Oscillator](#)
- [AFL Example – Enhanced](#)
- [Against all odds](#)
- [Alpha and Beta and R\\_Squared Indicator](#)
- [Andrews PitchforkV3.3](#)
- [Auto–Optimization Framework](#)
- [Automatic Trend–line](#)
- [BMTRIX Intermediate Term Market Trend Indicator](#)
- [Bollinger band normalization](#)
- [CCT Coppock Curve](#)
- [Chaikin's Volatility](#)
- [Compare Sectors against Tickers](#)
- [Coppock Curve](#)
- [Customised Avg. Profit %, Avg. Loss % etc](#)
- [Dave Landry PullBack Scan](#)
- [Demand Index](#)
- [DMI Spread Index](#)
- [Elder safe Zone Long + short](#)
- [Elder Triple Screen Trading System.](#)
- [End Of Year Trading](#)
- [Fibonacci Moving averages](#)
- [Fund Screener](#)
- [Futures – Dollar Move Today Indicator](#)
- [Know Sure Thing](#)
- [LunarPhase](#)
- [Modified Momentum Finder DDT–NB](#)
- [Monthly Coppock Guide](#)
- [Negative ROC Exporation](#)
- [nifty](#)
- [Option Calls, Puts and days till third friday.](#)
- [Polarized Fractal Efficiency](#)

- [Position Sizing and Risk Price Graph](#)
- [Position Sizing and Risk Price Graph – 2](#)
- [Positive ROC Exploration](#)
- [prakash](#)
- [RSI "based" Trading System](#)
- [SAR–ForNextBarStop](#)
- [Schiff Lines](#)
- [SectorRSI](#)
- [Shares To Buy Price Graph](#)
- [SIROC Momentum](#)
- [Smoothed RSI Buy Signals](#)
- [Trend Detection](#)
- [Varexlist](#)
- [Volume – compared with Moving Avg \(100%\)](#)
- [William's % R](#)
- [Williams %R Exploration](#)
- [Zig Zag Indicator with Valid Entry and Exit Points](#)

**More information:**

[Updated on–line reference](#)

**ROUND****Math functions****– round number to nearest integer**

**SYNTAX**      *round( NUMBER )*  
                  *round( ARRAY )*

**RETURNS**    NUMBER,  
                  ARRAY

**FUNCTION**    Rounds NUMBER or ARRAY to the nearest integer.

**EXAMPLE**     The formula "round( +11.5 )" returns +12. The formula "round( –11.4 )" returns –11.

**SEE ALSO**     The ceil() function; the floor() function; the int() function.

**References:**

The **ROUND** function is used in the following formulas in AFL on–line library:

- ['D/9E H'](#)
- ['D/9E H'](#)
- [Adaptive Price Channel](#)
- [AR\\_Prediction.afl](#)
- [Bullish Percent Index 2 files combined](#)
- [CCI Woodies Style](#)
- [Dave Landry PullBack Scan](#)
- [Ed Seykota's TSP: EMA Crossover System](#)
- [Ed Seykota's TSP: Support and Resistance](#)
- [Elder Triple Screen Trading System.](#)
- [Historical Volatility Scan – 50 Day](#)
- [Hurst "Like" DE](#)
- [Kagi Chart](#)
- [PFChart – High/Low prices Sept2003](#)
- [PF Chart – Close – April 2004](#)
- [Price with Woodies Pivots](#)
- [Random Walk](#)
- [Triangle exploration using PFChart](#)
- [Triangular Moving Average](#)
- [Triangular Moving Average new](#)

**More information:**

[Updated on–line reference](#)



**RSI****Indicators****– relative strength index**

**SYNTAX**     *RSI( periods = 14 )*  
                  *RSIa( array, periods = 14 )*

**RETURNS**    ARRAY

**FUNCTION**    Calculates the RSI indicator using *periods* range  
                  Second version RSIa accepts input array so it RSI can be applied to other arrays than close.

**EXAMPLE**     RSI( 12 )  
                  RSIa( High, 12 );

**SEE ALSO****Comments:**

<p><b>Tomasz Janeczko</b>  tj -- at -- amibroker.com  2007-07-27 09:34:56</p>	<pre>// Internally RSI is implemented as follows // function BuiltInRSIEquivalent( period ) {     P = N = 0;      result = Null;      for( i = 1; i &lt; BarCount; i++ )     {         diff = C[ i ] - C[ i - 1 ];         W = S = 0;         if( diff &gt; 0 ) W = diff;         if( diff &lt; 0 ) S = -diff;          P = ( ( period - 1 ) * P + W ) / period;         N = ( ( period - 1 ) * N + S ) / period;          if( i &gt;= period )             result[ i ] = 100 * P / ( P + N );     }     return result; }  Plot( BuiltInRSIEquivalent( 14 ), "RSI 1", colorRed ); Plot( RSI( 14 ), "RSI 2", colorBlue );</pre>
---	--

**References:**

The **RSI** function is used in the following formulas in AFL on-line library:

- [Adaptave Zones O/B &O/S Oscillator](#)
- [Against all odds](#)
- [Bollinger band normalization](#)
- [Bottom Trader](#)

- [CCT StochasticRSI](#)
- [colored CCI](#)
- [COMBO](#)
- [Compare Sectors against Tickers](#)
- [Derivative Oscillator](#)
- [Divergence indicator](#)
- [DT Oscillator](#)
- [Ehlers Laguerre RSI](#)
- [Follow the Leader](#)
- [Fund Screener](#)
- [MACD and histogram divergence detection](#)
- [MultiCycle 1.0](#)
- [Negative ROC Exporation](#)
- [Overbought issues, Oversold issues](#)
- [Positive ROC Exploration](#)
- [Ranking and sorting stocks](#)
- [Relative Strength Index](#)
- [RSI "based" Trading System](#)
- [RSI Double–Bottom](#)
- [RSI Pointer](#)
- [RSI Trendlines and Wedges](#)
- [RSIS](#)
- [SectorRSI](#)
- [Smoothed RSI Buy Signals](#)
- [Support Resistance levels](#)
- [Varexlist](#)
- [Volatility System](#)
- [Williams %R Exploration](#)

**More information:**

[Updated on–line reference](#)

**RWI****Indicators****– random walk index**

**SYNTAX**     *rwi( minperiods, maxperiods )*

**RETURNS**     ARRAY

**FUNCTION**     Calculates the Random Walk Index indicator as a difference between Random Walk Index from Highs (RWIHI() function) and Random Walk Index from Lows (RWILO() function).

**EXAMPLE**     *rwi( 9, 40 );*

**SEE ALSO****References:**

The **RWI** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**RWIHI****Indicators****– random walk index of highs****SYNTAX**     *rwihl( minperiods, maxperiods )***RETURNS**     ARRAY**FUNCTION**     Calculates the Random Walk Index from Highs.**EXAMPLE**     *rwihl( 9, 40 );***SEE ALSO****References:**

The **RWIHI** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**RWLO****Indicators****– random walk index of lows****SYNTAX**     *rwlo( minperiods, maxperiods )***RETURNS**     ARRAY**FUNCTION**     Calculates the Random Walk Index from Lows.**EXAMPLE**     *rwlo( 9, 40 );***SEE ALSO****References:**

The **RWLO** function is used in the following formulas in AFL on–line library:

**More information:**[Updated on–line reference](#)

## **SAR**

### **– parabolic stop–and–reverse**

**Indicators**  
(AFL 1.3)

**SYNTAX**     *sar( accel = 0.02, max = 0.2 )*

**RETURNS**     ARRAY

**FUNCTION**     Calculates Parabolic SAR indicator. Acceleration is given by accel argument and maximum acceleration level is given by max argument

**EXAMPLE**     sar()

**SEE ALSO**

#### **References:**

The **SAR** function is used in the following formulas in AFL on–line library:

- [EKEKO SAR–MF](#)
- [Main price chart with Rainbow &SAR](#)
- [Parabolic SAR in VBScript](#)
- [Trend Analysis\\_Comentary](#)
- [Volatility Breakout with Bollinger Bands](#)

#### **More information:**

[Updated on–line reference](#)

**SAY****– speaks provided text****Miscellaneous functions**  
(AFL 2.90)**SYNTAX**     **Say("text")****RETURNS**     NOTHING**FUNCTION**     Say() function speaks user–specified text (Windows XP, on lower–end Windows you need to install Microsoft Speech API, voice settings are in Windows Control Panel)

**EXAMPLE**     `// simple example`  
                  `Say("Testing text to speech engine");`

`// helpful helper functions`

```

function SayOnce( text )
{
    if( StaticVarGetText("lastsaidtext") != text )
    {
        Say( text );
        StaticVarSetText("lastsaidtext", text );
    }
}

function SayNotTooOften( text, Minperiod )
{
    elapsed=GetPerformanceCounter()/1000;
    Lastelapsed = Nz( StaticVarGet("lastsaytime") );

    if( elapsed - Lastelapsed > Minperiod )
    {
        StaticVarSet("lastsaytime", elapsed );

        Say( text );
    }
}

SayOnce("Testing "+Name() );
SayNotTooOften( "Say not more often than every 60 seconds", 60 );

```

**SEE ALSO****References:**

The **Say** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on-line reference](#)



**SECOND**  
**– get current bar's second****Date/Time**  
(AFL 2.0)**SYNTAX**     *second()***RETURNS**    ARRAY**FUNCTION**    Retrieves current bar's second**EXAMPLE**     Hour()\*10000 + Minute() \* 100 + Second()**SEE ALSO**     Hour(), Minute(), TimeNum()**References:**

The **SECOND** function is used in the following formulas in AFL on–line library:

- [Export Intraday Data](#)

**More information:**

[Updated on–line reference](#)

**SECTORID****Information / Categories****– get sector ID / name**

(AFL 1.8)

**SYNTAX**      *sectorid( mode = 0 )***RETURNS**    NUMBER/STRING

**FUNCTION**    Retrieves current stock sector ID/name When mode = 0 (the default value ) this function returns numerical sector ID (consecutive sector number)  
When mode = 1 this function returns name of the sector.

**EXAMPLE**     Filter = SectorID() == 7 OR SectorID() == 9;  
AddTextColumn( SectorID( 1 ), "Sector name" );

**SEE ALSO****References:**

The **SECTORID** function is used in the following formulas in AFL on–line library:

- [Compare Sectors against Tickers](#)
- [MACD and histogram divergence detection](#)
- [Steve Woods' Cum. Vol. Float + Cum. Vol. Channels](#)
- [Steve Woods' Cumulative Vol. Percentage Indicator](#)

**More information:**

[Updated on–line reference](#)

**SELECTEDVALUE****– retrieves value of the array at currently selected date/time point****Exploration /  
Indicators**  
(AFL 2.1)**SYNTAX**      *selectedvalue( ARRAY )***RETURNS**    NUMBER**FUNCTION**    Retrieves array value at currently selected bar. Main purpose: commentary and interpretation code.**EXAMPLE**      selectedvalue( close )**SEE ALSO****Comments:**

<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2003-05-11 06:02:38	<p>SelectedValue(array) is a function that retrieves 'selected element' of the array. Since 'selection line' is available only for CHARTS. SelectedValue gives the value of array at bar that is currently selected in chart by vertical line. This is how it works in INDICATORS, INTERPRETATION and CHART COMMENTARY (because they are relative to selected bar)</p> <p>In AA window 'selected element' means THE LAST BAR of currently selected analysis range. It is the last available bar for "all quotes" and "last n quotes" range. It is the the bar corresponding to "End Date" when using "From-To" range.</p> <p>So if you choose range: "all quotes" in AA  SelectedValue function is equivalent to</p> <p>array[ BarCount – 1 ]</p>
---	---

**References:**

The **SELECTEDVALUE** function is used in the following formulas in AFL on-line library:

- [Andrews Pitchfork](#)
- [Andrews PitchforkV3.3](#)
- [AR\\_Prediction.afl](#)
- [Bull/Bear Volume](#)
- [Candle Identification](#)
- [Candle Stick Demo](#)
- [CCI\(20\) Divergence Indicator](#)
- [Chandelier Exit](#)
- [Color Display.afl](#)
- [DateNum\\_DateStr](#)
- [Dave Landry PullBack Scan](#)
- [ekeko price chart](#)
- [Elder safe Zone Long + short](#)
- [Elder Triple Screen Trading System.](#)
- [Fibonacci Moving averages](#)
- [Gordon Rose](#)

- [Hurst "Like" DE](#)
- [Luna Phase](#)
- [LunarPhase](#)
- [Multiple sinus noised](#)
- [nifty](#)
- [Option Calls, Puts and days till third friday.](#)
- [PF Chart – Close – April 2004](#)
- [prakash](#)
- [SAR–ForNextBarStop](#)
- [Schiff Lines](#)
- [tomy\\_frenchy](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [Zig Zag Indicator with Valid Entry and Exit Points](#)

**More information:**

[Updated on–line reference](#)

**SETBACKTESTMODE**

Trading system toolbox

**– Sets working mode of the backtester**

(AFL 3.0)

**SYNTAX**     *SetBacktestMode( mode )***RETURNS**    NOTHING**FUNCTION**    Sets working mode of the backtester. A 'mode' parameter is one of the following backtest modes:

Supported backtest modes:

- **backtestRegular** – regular, signal-based backtest, redundant signals are removed as shown in [this picture](#)
- **backtestRegularRaw** – signal-based backtest, redundant (raw) signals are NOT removed, only one position per symbol allowed
- **backtestRegularRawMulti** – signal-based backtest, redundant (raw) signals are NOT removed, MULTIPLE positions per symbol will be open if BUY/SHORT signal is "true" for more than one bar and there are free funds, Sell/Cover exit all open positions on given symbol, Scale-In/Out work on all open positions of given symbol at once.
- **backtestRotational** – rotational trading system see [this](#).

**EXAMPLE**     *// default, as in 4.90, regular, signal-based backtest, redundant signals are removed*  
                   *SetBacktestMode( backtestRegular );*

*// signal-based backtest, redundant (raw) signals are NOT removed, only one position per symbol allowed*  
                   *SetBacktestMode( backtestRegularRaw );*

*// signal-based backtest, redundant (raw) signals are NOT removed, MULTIPLE positions per symbol will be open if BUY/SHORT signal is "true" for more than one bar and there are free funds*  
*// Sell/Cover exit all open positions on given symbol, Scale-In/Out work on all open positions of given symbol at once.*  
                   *SetBacktestMode( backtestRegularRawMulti );*

*// rotational trading mode - equivalent of EnableRotationalTrading() call*  
                   *SetBacktestMode( backtestRotational );*

**SEE ALSO**     [EnableRotationalTrading\(\)](#) function**References:**The **SetBacktestMode** function is used in the following formulas in AFL on-line library:**More information:**[Updated on-line reference](#)

**SETBARSREQUIRED**

– set number of previous and future bars needed for script/DLL to properly execute

Miscellaneous  
functions  
(AFL 2.1)

**SYNTAX**      *setbarsrequired( backwardref = -1, forwardref = -1 )*

**RETURNS**    nothing

**FUNCTION**    set number of previous and future bars needed for script/DLL to properly execute. If your formula is pure AFL you don't need to use this function at all, as AmiBroker automatically calculates number of bars required for all its built-in functions. But if you are using script or a DLL you may need to use this function to make sure that your indicators are properly calculated in QuickAFL mode. Specifying -1 means no change. For example if you are using the script that calculates 100 bar moving average you may need to call SetBarsRequired( 100, 0 ); at the very beginning of your formula. Please note that in most cases it is not necessary (even if you are using script or DLL) because AmiBroker always provides at least 30 past data bars more than needed.

**EXAMPLE**      *setbarsrequired( 100000, 100000 ) // require all past and all future bars*

**SEE ALSO****References:**

The **SETBARSREQUIRED** function is used in the following formulas in AFL on-line library:

- [Adaptive Laguerre Filter, from John Ehlers](#)
- [AFL Example](#)
- [AFL Example – Enhanced](#)
- [Andrews Pitchfork](#)
- [Andrews PitchforkV3.3](#)
- [AR\\_Prediction.afl](#)
- [Bullish Percent Index 2 files combined](#)
- [Bullish Percent Index 2004](#)
- [Darvas Box](#)
- [DateNum\\_DateStr](#)
- [Ehlers Center of Gravity Oscillator](#)
- [Ehlers CyberCycle](#)
- [Ehlers Dominant Cycle Period](#)
- [Ehlers Fisher Transform](#)
- [Ehlers Instantaneous Trend](#)
- [Ehlers Laguerre RSI](#)
- [Kagi Chart](#)
- [Modified Darvas Box](#)
- [Monte Carlo](#)
- [Monthly bar chart](#)
- [Multiple sinus noised](#)
- [PFChart – High/Low prices Sept2003](#)
- [PFchart with range box sizes](#)
- [PF Chart – Close – April 2004](#)
- [Random Walk](#)
- [Renko Chart](#)

- [Schiff Lines](#)
- [Three Line Break – TLB](#)
- [tomy\\_frenchy](#)
- [Triangle exploration using PFChart](#)
- [Triangular Moving Average new](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [Volatility System](#)

**More information:**

[Updated on–line reference](#)

## SETCARTBKCOLOR

– set background color of a chart

**Indicators**  
(AFL 2.80)

**SYNTAX**     *SetChartBkColor( color )*

**RETURNS**    NOTHING

**FUNCTION**   Sets chart background to user-specified color

**EXAMPLE**    SetChartBkColor( colorBlue );

**SEE ALSO**    [ColorHSB\(\)](#) function , [ColorRGB\(\)](#) function

**References:**

The **SetChartBkColor** function is used in the following formulas in AFL on-line library:

- [Hull Moving Average](#)
- [SUPER PIVOT POINTS](#)

**More information:**

[Updated on-line reference](#)



**SETCHARTBKGRADIENTFILL****Indicators**  
(AFL 2.90)**– enables background gradient color fill in indicators****SYNTAX**     *SetChartBkGradientFill( topcolor, bottomcolor, titlebkcolor = default )***RETURNS**    NOTHING**FUNCTION**    Enables background gradient color fill in indicators.

Please note that this is independent from chart background color (background color fills entire pane, gradient fill is only for actual chart interior, so axes area is not affected by gradient fill)

- topcolor – specifies top color of the gradient fill
- bottomcolor – specifies bottom color of the gradient fill
- titlebkcolor – (optional) the background color of title text. If not specified then top color is
- automatically used for title background.

**EXAMPLE**     `SetChartBkGradientFill( ParamColor( "BgTop" ,  
colorWhite) , ParamColor( "BgBottom" , colorLightYellow) ) ;`

**SEE ALSO**     [PLOT\(\)](#) function , [PLOTFOREIGN\(\)](#) function , [PLOTGRID\(\)](#) function , [PLOTOLHC\(\)](#) function ,  
[PLOTSHAPES\(\)](#) function , [PlotText\(\)](#) function , [PLOTVAPOVERLAY\(\)](#) function

**References:**

The **SetChartBkGradientFill** function is used in the following formulas in AFL on–line library:

- [LunarPhase](#)

**More information:**

[Updated on–line reference](#)

**SETOPTIONS****Exploration / Indicators****– set/clear/overwrite defaults for chart pane options**

(AFL 2.70)

**SYNTAX**     **SetChartOptions( Mode = 0, Flags = 0, gridFlags = chartGridMiddle )****RETURNS**     NOTHING**FUNCTION**     Allows to set/clear/overwrite/set defaults for chart pane options

- Mode – specifies how options are set:
  - ◆ 0 – set only the DEFAULT values for new chart. Defaults are applied only once when chart is inserted in a new pane, so later you can modify any option using Indicator Builder
  - ◆ 1 – overwrite – the values specified in 2nd and 3rd argument overwrite any previously set values
  - ◆ 2 – set flag – flags specified in 2nd and 3rd parameter are binary-ORed with the current values, so effectively these options are set while remaining are unchanged
  - ◆ 3 – reset flag – flags specified in 2nd and 3rd parameter are cleared while the others remain unchanged.
- Flags – allowable flags are:  
chartShowDates, chartLogarithmic, chartShowArrows, chartWrapTitle (4.75 or higher)
- gridFlags – (for internal AmiBroker use – do not use it in your own coding as this parameter will be eventually removed) allowable values are: chartGridDiv100, chartGridPercent, chartGridDiv1000, chartGridMargins chartGridMiddle, chartGrid0, chartGrid30, chartGrid70, chartGrid10, chartGrid90, chartGrid50, chartGrid100, chartGrid20, chartGrid80, chartGrid1

**EXAMPLE**     *//to mark "Show arrows" by default in a new chart use*  
                  `SetChartOptions( 0, chartShowArrows );`

Example 2 (works only with version 4.75 or higher):

```
SetChartOptions(2, chartWrapTitle );
Title="this is a test of automatic wrapping of title text that is
too long to fit in single line, for that reason this sample formula
uses very long text. I hope you are enjoying the sample";
```

**SEE ALSO****References:**

The **SetChartOptions** function is used in the following formulas in AFL on–line library:

- [Dave Landry PullBack Scan](#)
- [Elder Triple Screen Trading System.](#)
- [Fibonacci Moving averages](#)
- [LunarPhase](#)
- [Market Profile &Market Volume Profile](#)
- [nifty](#)
- [prakash](#)
- [ZigZag filter rewritten from scratch in AFL](#)

**More information:**

[Updated on-line reference](#)

**SETCUSTOMBACKTESTPROC**

Trading system toolbox

– define custom backtest procedure formula file

(AFL 2.70)

**SYNTAX**     *SetCustomBacktestProc( filename, enable = True )***RETURNS**    NOTHING**FUNCTION**   This function allows changing custom backtest procedure file from AFL formula level.To learn more about custom backtester procedures please read [this document](#).

Parameters

- *filename* parameter instructs backtester to use external formula file as custom backtest procedure, if empty – it means use current formula
- *enable* = True (default) – enables custom backtesting procedure (the same as `SetOption("UseCustomBacktestProc", True );`  
*enable* = False – disables custom proc

**EXAMPLE**     `SetCustomBacktestProc( "Formulas\MyCustomBacktest.afl", True );`**SEE ALSO****References:**The **SetCustomBacktestProc** function is used in the following formulas in AFL on–line library:

- [Customised Avg. Profit %, Avg. Loss % etc](#)

**More information:**[Updated on–line reference](#)

**SETFOREIGN**

– replace current price arrays with those of foreign security

Referencing other symbol data  
(AFL 2.5)

**SYNTAX**      **SetForeign( ticker, fixup = True, tradeprices = False )**

**RETURNS**      NUMBER

**FUNCTION**      The **SetForeign** function replaces current price/volume arrays with those of foreign security, returns True (1) if ticker exists, False (0) otherwise.

If ticker does not exist (and function returns false) price arrays are not changed at all.

*fixup* parameter controls if data holes are filled with previous bar data or not. If *fixup* is False then data holes are filled with Null value. For detailed discussion on *fixup* parameter please check **Foreign** function.

*tradeprices* parameter controls if trade price arrays should be replaced too. If it is set to TRUE, then not only OHLC, V, OI, Avg arrays are set to foreign symbol values, but also BuyPrice, SellPrice, ShortPrice, CoverPrice, PointValue, TickSize, RoundLotSize, MarginDeposit variables are set to correspond to foreign security. This allows Equity() to work well with SetForeign.

Single **SetForeign( "ticker" )** call is equivalent to the following sequence:

```
C = Foreign( "ticker", "C" );
O = Foreign( "ticker", "O" );
H = Foreign( "ticker", "H" );
L = Foreign( "ticker", "L" );
V = Foreign( "ticker", "V" );
OI = Foreign( "ticker", "I" );
Avg = ( C + H + L )/3;
```

but 6x faster (SetForeign takes about the same time as single foreign). To restore original prices call **RestorePriceArrays()**

**EXAMPLE**      *// Example 1: Plot the indicator using foreign security data*  
                  *SetForeign( "MSFT" );*  
                  *Plot( Ultimate(), "Ultimate from MSFT", colorRed );*  
                  *RestorePriceArrays();*

*// Example 2: Use SetForeign with Equity function*  
                  *SetForeign( "MSFT", True, True );*  
                  *Buy = Cross( MACD(), Signal() );*  
                  *Sell = Cross( Signal(), MACD() );*  
                  *e = Equity(); // backtest on MSFT*  
                  *RestorePriceArrays( True ); //*

**SEE ALSO**      **FOREIGN()** function , **RestorePriceArrays()** function

**References:**

The **SetForeign** function is used in the following formulas in AFL on–line library:

- [Ranking and sorting stocks](#)
- [Ranking Ticker WatchList](#)
- [WLBuildProcess](#)

**More information:**

[Updated on–line reference](#)

**SETFORMULANAME**

Trading system toolbox

– set the name of the formula

(AFL 2.5)

**SYNTAX**     *SetFormulaName( string )***RETURNS**    NOTHING**FUNCTION**    Allows to programatically change the name of the formula that is displayed in the backtest result explorer.**EXAMPLE**     `SetFormulaName( "My Holy Grail System" );`**SEE ALSO****References:**

The **SetFormulaName** function is used in the following formulas in AFL on–line library:

- [AFL Example](#)
- [AFL Example – Enhanced](#)
- [Triangular Moving Average new](#)

**More information:**

[Updated on–line reference](#)

**SETOPTION**

Trading system toolbox

– sets options in automatic analysis settings

(AFL 2.3)

**SYNTAX**     *SetOption( field, value )***RETURNS**     NOTHING**FUNCTION**     Sets various options in automatic analysis settings. Affects also Equity() function results.*field* – is a string that defines the option to change. There are following options available:

- "NoDefaultColumns" – if set to True – exploration does not have default Ticker and Date/Time columns
- "InitialEquity"
- "AllowSameBarExit"
- "ActivateStopsImmediately"
- "AllowPositionShrinking"
- "FuturesMode"
- "InterestRate"
- "MaxOpenPositions" – maximum number of simultaneously open positions (trades) in portfolio backtest/optimization
- "WorstRankHeld" – the worst rank of symbol to be held in rotational trading mode (see **EnableRotationalTrading** for more details)
- "MinShares" – the minimum number of shares required to open the position in the backtester/optimizer. If you don't have enough funds to purchase that many, trade will NOT be entered
- "MinPosValue" – (4.70.3 and above) the minimum dollar amount required to open the position in the backtester/optimizer. If you don't have enough funds trade will NOT be entered
- "PriceBoundChecking" – if set to False – disables checking and adjusting buyprice/sellprice/coverprice/shortprice arrays to current symbol High–Low range.
- CommissionMode –
  - 0 – use portfolio manager commission table
  - 1 – percent of trade
  - 2 – \$ per trade
  - 3 – \$ per share/contract
- CommissionAmount – amount of commission in modes 1..3
- AccountMargin (in old versios it was 'MarginRequirement') – account margin requirement (as in settings), 100 = no margin
- ReverseSignalForcesExit – reverse entry signal forces exit of existing trade (default = True )
- UsePrevBarEquityForPosSizing – Affects how percent of current equity position sizing is performed.  
False (default value) means: use current (intraday) equity to perform position sizing,  
True means: use previous bar closing equity to perform position sizing
- PortfolioReportMode – sets backtester report mode:
  - 0 – trade list
  - 1 – detailed log
  - 2 – summary
  - 3 – no output (custom only)



- UseCustomBacktestProc – True/False – allows to turn on/off custom backtest procedure
- EveryBarNullCheck – allows to turn on checking for Nulls in arithmetic operations on every bar in the array (by default it is OFF – i.e. AmiBroker checks for nulls that appear in the beginning of the array and in the end of the array and once non-null value is detected it assumes no further holes (nulls) in the middle). Turning "EveryBarNullCheck" to True allows to extend these checks to each and every bar which is the way 4.74.x and earlier versions worked.  
Note however that turning it on gives huge performance penalty (arithmetic operations are performed even 4x slower when this option is ON, so don't use it unless you really have to).
- HoldMinBars – Number – if set to value > 0 – it disables exit during user-specified number of bars even if signals/stops are generated during that period
- EarlyExitBars – Number if set to value > 0 – causes that special early exit (redemption) fee is charged if trade is exited during this period
- EarlyExitFee – defines the % (percent) value of early exit fee
- HoldMinDays – Number – if set to value > 0 – it disables exit during user-specified number of CALENDAR DAYS (not bars) even if signals/stops are generated during that period
- EarlyExitDays – Number if set to value > 0 – causes that special early exit (redemption) fee is charged if trade is exited during the period specified in calendar days (not bars).
- DisableRuinStop – if set to TRUE built-in ruin stop is disabled

**WARNING:** If you change the option on *\*per-symbol\** basis the composite results (%profit for example) will be **DISTORTED** since calculations assume that **OPTIONS** are constant for all symbols in one backtest run. 'HoldMinBars', 'EarlyExit...' options are exception from this rule (i.e. can be safely set on per-symbol basis)

**EXAMPLE**

```

SetOption("InitialEquity", 5000 );
SetOption("AllowPositionShrinking", True );
SetOption("MaxOpenPositions", 5 );
PositionSize = -100/5;

```

**SEE ALSO** [EnableRotationalTrading\(\)](#) function , [EQUITY\(\)](#) function

#### References:

The **SetOption** function is used in the following formulas in AFL on-line library:

- [AFL Example](#)
- [AFL Example – Enhanced](#)
- [Black Scholes Option Pricing](#)
- [Ed Seykota's TSP: EMA Crossover System](#)
- [Ed Seykota's TSP: Support and Resistance](#)
- [End Of Year Trading](#)
- [Ranking and sorting stocks](#)
- [Relative Strength](#)
- [RUTVOL timing signal with BB Scoring routine](#)
- [Simple Candle Exploration](#)
- [Triangular Moving Average new](#)
- [Visualization of stoplosses and profit in chart](#)
- [Volatility System](#)

**More information:**

[Updated on-line reference](#)

**SETPOSITIONSIZE**

Trading system toolbox

– set trade size

(AFL 2.70)

**SYNTAX**     *SetPositionSize( size, method )***RETURNS**     ARRAY**FUNCTION**     This function allows to control trade (position) size in four different ways, depending on 'method' parameter.

Parameters:

*size* (ARRAY) defines desired trade size*method* (ARRAY) defines how 'size' is interpreted

- *spsValue* (=1) – dollar value of size (as in previous versions)
- *spsPercentOfEquity* (=2) – size expressed as percent of portfolio–level equity (size must be from ..100 (for regular accounts) or .1000 for margin accounts)
- *spsShares* (=4) – size expressed in shares/contracts (size must be > 0 )
- *spsPercentOfPosition* (=3) – size expressed as percent of currently open position (for SCALING IN and SCALING OUT ONLY)
- *spsNoChange* (=0) – don't change previously set size for given bar

New SetPositionSize function automatically encodes new methods of expressing position size into old "positionsize" variable as follows:

- values below –2000 encode share count,
- values between –2000 and –1000 encode % of current position
- values between –1000 and 0 encode % of portfolio equity
- values above 0 encode dollar value

Although it is possible to assign these values directly to old–style PositionSize variable, new code should use SetPositionSize function for clarity.

**EXAMPLE**     For example to liquidate 50% of position simply use

```
SetPositionSize( 50, spsPercentOfPosition * ( Buy == sigScaleOut )
);
```

Special value *spsNoChange* (=0) means don't change previously set size for given bar (allows to write constructs like that):

```
SetPositionSize( 100, spsShares ); // 100 shares by default
SetPositionSize( 50, IIf( Buy == sigScaleOut, spsPercentOfPosition,
spsNoChange ) ); // for scale-out use 50% of current position size
```

Example of code that exits 50% on first profit target, 50% on next profit target and everything at trailing stop:

```
Buy = Cross( MA( C, 10 ), MA( C, 50 ) );
```

```

Sell = 0;

// the system will exit
// 50% of position if FIRST PROFIT TARGET stop is hit
// 50% of position is SECOND PROFIT TARGET stop is hit
// 100% of position if TRAILING STOP is hit

FirstProfitTarget = 10; // profit
SecondProfitTarget = 20; // in percent
TrailingStop = 10; // also in percent

priceatbuy=0;
highsincebuy = 0;

exit = 0;

for( i = 0; i < BarCount; i++ )
{
    if( priceatbuy == 0 AND Buy[ i ] )
    {
        priceatbuy = BuyPrice[ i ];
    }

    if( priceatbuy > 0 )
    {
        highsincebuy = Max( High[ i ], highsincebuy );

        if( exit == 0 AND
            High[ i ] >= ( 1 + FirstProfitTarget * 0.01 ) * priceatbuy
        )
        {
            // first profit target hit - scale-out
            exit = 1;
            Buy[ i ] = sigScaleOut;
        }

        if( exit == 1 AND
            High[ i ] >= ( 1 + SecondProfitTarget * 0.01 ) *
priceatbuy )
        {
            // second profit target hit - exit
            exit = 2;
            SellPrice[ i ] = Max( Open[ i ], ( 1 + SecondProfitTarget *
0.01 ) * priceatbuy );
        }

        if( Low[ i ] <= ( 1 - TrailingStop * 0.01 ) * highsincebuy )
        {
            // trailing stop hit - exit
            exit = 3;
            SellPrice[ i ] = Min( Open[ i ], ( 1 - TrailingStop * 0.01

```

```

    ) * highsincebuy );
    }

    if( exit >= 2 )
    {
        Buy[ i ] = 0;
        Sell[ i ] = exit + 1; // mark appropriate exit code
        exit = 0;
        priceatbuy = 0; // reset price
        highsincebuy = 0;
    }
}

SetPositionSize( 50, spsPercentOfEquity );
SetPositionSize( 50, spsPercentOfPosition * ( Buy == sigScaleOut )
); // scale out 50% of position

```

**SEE ALSO****References:**

The **SetPositionSize** function is used in the following formulas in AFL on-line library:

- [Customised Avg. Profit %, Avg. Loss % etc](#)
- [Ed Seykota's TSP: EMA Crossover System](#)

**More information:**

[Updated on-line reference](#)

**SETSORTCOLUMNS****Exploration /  
Indicators**  
(AFL 2.90)**– sets the columns which will be used for sorting in AA window****SYNTAX**      *SetSortColumns( col1, col2, .... )***RETURNS**    NOTHING

**FUNCTION**    sets the columns which will be used for sorting. col1, col2, ... col10 –Column numbers are ONE-based. Positive number means sort ASCENDING, negative number means sort DESCENDING. Upto 10 columns can be specified for multiple-column sort. Each subsequent call to SetSortColumns overwrites previous one.

**EXAMPLE**    *// sort by 5th column in ascending order*  
*SetSortColumns( 5 )*

*// sort by 3rd column in descending order*  
*SetSortColumns( -3 )*

*// sort by 1st column in ascending order AND then by Second column*  
*in descending order (multiple-column sort).*  
*SetSortColumns( 1, -2 );*

**SEE ALSO****References:**

The **SetSortColumns** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)

**SETTRADEDELAYS**

Trading system toolbox

**– allows to control trade delays applied by the backtester**

(AFL 2.1)

**SYNTAX**      *settradedelays( buydelay, selldelay, shortdelay, coverdelay )***RETURNS**      nothing**FUNCTION**      Sets trade delays applied by the backtester. This function allows you to override trade delays from the "Settings" page.**EXAMPLE**      *settradedelays( 1, 1, 1, 1 )***SEE ALSO****References:**

The **SETTRADEDELAYS** function is used in the following formulas in AFL on–line library:

- [AFL Example](#)
- [AFL Example – Enhanced](#)
- [Ed Seykota's TSP: EMA Crossover System](#)
- [Ed Seykota's TSP: Support and Resistance](#)
- [Evaluating Candle Patterns in a trading system](#)
- [RUTVOL timing signal with BB Scoring routine](#)
- [Triangular Moving Average new](#)
- [Volatility System](#)

**More information:**[Updated on–line reference](#)

**SIGN****Math functions****– returns the sign of the number/array**

(AFL 2.50)

**SYNTAX**     *sign( x )***RETURNS**     ARRAY or NUMBER**FUNCTION**     Sign function returns 1 if x value is greater than zero, –1 if the x is less than zero and 0 if x equals zero. x can be a number or array.**EXAMPLE****SEE ALSO****References:**

The **sign** function is used in the following formulas in AFL on–line library:

- [Elder Triple Screen Trading System](#).

**More information:**

[Updated on–line reference](#)



**SIGNAL****Indicators****– macd signal line**

**SYNTAX**      *signal( fast = 12, slow = 26, signal = 9 )*

**RETURNS**     ARRAY

**FUNCTION**    Calculates the Signal line of MACD indicator.

**EXAMPLE**     *signal( 14, 28, 10 );*

**SEE ALSO****References:**

The **SIGNAL** function is used in the following formulas in AFL on–line library:

- [AFL Example – Enhanced](#)
- [Bollinger band normalization](#)
- [Compare Sectors against Tickers](#)
- [Customised Avg. Profit %, Avg. Loss % etc](#)
- [Dinapoli Guru Commentary](#)
- [Ed Seykota's TSP: EMA Crossover System](#)
- [Ed Seykota's TSP: Support and Resistance](#)
- [ekeko price chart](#)
- [Elder Impulse Indicator](#)
- [Elder Impulse Indicator V2](#)
- [Elder Triple Screen Trading System.](#)
- [Fund Screener](#)
- [hassan](#)
- [Indicator Explorer \(ZigZag\)](#)
- [MACD and histogram divergence detection](#)
- [MACD commentary](#)
- [MACD Histogram – Change in Direction](#)
- [MACD indicator display](#)
- [MACD optimize](#)
- [ROC of MACD Weekly](#)
- [STO &MACD Buy Signals with Money–Management](#)
- [The Mean RSIt](#)
- [The Mean RSIt \(variations\)](#)
- [Trending or Trading](#)
- [Trending Ribbon](#)
- [Varexlist](#)

**More information:**

[Updated on–line reference](#)

**SIN****Math functions****– sine function**

**SYNTAX**     *sin( NUMBER )*  
                 *sin( ARRAY )*

**RETURNS**    NUMBER,  
                 ARRAY

**FUNCTION**   Returns the sine of NUMBER or ARRAY. This function assumes that the ARRAY values are in radians.

**EXAMPLE**    You can plot a sine wave using the formula "sin(cum(0.05))." Increasing the value in this formula (i.e., "0.05") will increase the frequency of the sine wave.

**SEE ALSO**    The atan() function ; the cos() function.

**References:**

The **SIN** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [Color Display.afl](#)
- [Cycle Highlighter](#)
- [Cycle Highlighter \(auto best–fit\)](#)
- [Luna Phase](#)
- [Moving Average "Crash" Test](#)
- [Multiple sinus noised](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**

[Updated on–line reference](#)

## **SINH** – hyperbolic sine function

**Math functions**  
(AFL 2.80)

**SYNTAX**     *sinh( NUMBER )*  
                 *sinh( ARRAY )*

**RETURNS**    NUMBER,  
                 ARRAY

**FUNCTION**   Returns the hyperbolic sine of NUMBER or ARRAY. This function assumes that the ARRAY values are in radians.

### **EXAMPLE**

### **SEE ALSO**

### **References:**

The **sinh** function is used in the following formulas in AFL on–line library:

### **More information:**

[Updated on–line reference](#)

**SQRT****Math functions****– square root**

**SYNTAX**      *sqrt( NUMBER )*  
                 *sqrt( ARRAY )*

**RETURNS**    NUMBER,  
                 ARRAY

**FUNCTION**    Calculates the square root of NUMBER or ARRAY. The square root of a negative number always returns a zero result.

**EXAMPLE**     The formula "sqrt( 16 )" returns 4

**SEE ALSO****References:**

The **SQRT** function is used in the following formulas in AFL on–line library:

- ['R' Channel](#)
- [Black Scholes Option Pricing](#)
- [crMathLib](#)
- [Dave Landry PullBack Scan](#)
- [Elder Triple Screen Trading System.](#)
- [Historical Volatility Scan – 6/100](#)
- [Historical Volatility Scan – 50 Day](#)
- [Hull Moving Average](#)
- [Option Calls, Puts and days till third friday.](#)
- [Polarized Fractal Efficiency](#)
- [Probability Calculator](#)
- [RSI Trendlines and Wedges](#)
- [The Fibonacci behavior](#)

**More information:**

[Updated on–line reference](#)

**STATICVARGET****– gets the value of static variable****Miscellaneous functions**

(AFL 2.60)

**SYNTAX**      *StaticVarGet( "varname" )***RETURNS**     NUMBER or STRING**FUNCTION**    Gets the value of static variable.

*Static variable* – the variable has static duration (it is allocated when the program begins and deallocated when the program ends) and initializes it to Null unless another value is specified. Static variables allow to share values between various formulas. Only NUMBER and STRING static variables are supported now (not ARRAYS).

**EXAMPLE****SEE ALSO**    [StaticVarSet\(\)](#) function , [StaticVarSetText\(\)](#) function , [StaticVarGetText\(\)](#) function**References:**

The **StaticVarGet** function is used in the following formulas in AFL on–line library:

- [AFL Timing functions](#)

**More information:**[Updated on–line reference](#)

**STATICVARGETTEXT****Miscellaneous functions****– gets the value of static variable as string**

(AFL 2.60)

**SYNTAX**     *StaticVarGetText( "varname" )***RETURNS**    STRING

**FUNCTION**   Gets the value of static variable as string.  
 The only difference between StaticVarGet is that this function always returns string, so if given static variable does not exist it returns empty string "" instead of Null. Numbers are also converted to string.

*Static variable* – the variable has static duration (it is allocated when the program begins and deallocated when the program ends) and initializes it to Null unless another value is specified. Static variables allow to share values between various formulas. Only NUMBER and STRING static variables are supported now (not ARRAYS).

**EXAMPLE**     myvar = *StaticVarGetText*( "MyVariable" );

```

if( myvar == " " )
{
    printf( "Not Set" );
}
else
{
    printf( "Variable Set: " + myvar );
}
  
```

**SEE ALSO**     *StaticVarGet()* function , *StaticVarSet()* function , *StaticVarSetText()* function

**References:**

The **StaticVarGetText** function is used in the following formulas in AFL on–line library:

- [Calculate composites for tickers in list files](#)
- [DateNum\\_DateStr](#)
- [Ranking and sorting stocks](#)
- [TWS auto–export Executions–file parser](#)

**More information:**

[Updated on–line reference](#)

**STATICVARREMOVE**  
**– remove static variable****Miscellaneous functions**  
(AFL 2.80)**SYNTAX**     *StaticVarRemove( "variablename" )***RETURNS**    NOTHING**FUNCTION**    This function removes static variable and releases associated memory.**EXAMPLE****SEE ALSO**    [StaticVarGet\(\)](#) function , [StaticVarGetText\(\)](#) function , [StaticVarSet\(\)](#) function ,  
[StaticVarSetText\(\)](#) function**References:**

The **StaticVarRemove** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**STATICVARSET****– sets the value of static variable****Miscellaneous functions**  
(AFL 2.60)**SYNTAX**     *StaticVarSet( "varname", value )***RETURNS**     NUMBER**FUNCTION**     Sets the value of static variable. Returns 1 on success 0 on failure.

*Static variable* – the variable has static duration (it is allocated when the program begins and deallocated when the program ends) and initializes it to Null unless another value is specified. Static variables allow to share values between various formulas. Only NUMBER and STRING static variables are supported now (not ARRAYS).

**EXAMPLE****SEE ALSO**     [StaticVarSetText\(\)](#) function , [StaticVarGet\(\)](#) function**References:**

The **StaticVarSet** function is used in the following formulas in AFL on–line library:

- [AFL Timing functions](#)
- [Ranking and sorting stocks](#)

**More information:**[Updated on–line reference](#)



**STATICVARSETTEXT****Miscellaneous functions****– Sets the value of static string variable.**

(AFL 2.60)

**SYNTAX**     *StaticVarSetText( "varname", "value" )***RETURNS****FUNCTION**     Sets the value of static string variable. Returns 1 on success 0 on failure.

*Static variable* – the variable has static duration (it is allocated when the program begins and deallocated when the program ends) and initializes it to Null unless another value is specified. Static variables allow to share values between various formulas. Only NUMBER and STRING static variables are supported now (not ARRAYS).

**EXAMPLE****SEE ALSO**     [StaticVarSet\(\)](#) function , [StaticVarGet\(\)](#) function**References:**

The **StaticVarSetText** function is used in the following formulas in AFL on–line library:

- [Calculate composites for tickers in list files](#)
- [DateNum\\_DateStr](#)
- [Ranking and sorting stocks](#)
- [TWS auto–export Executions–file parser](#)

**More information:**[Updated on–line reference](#)

**STATUS****– get run–time AFL status information****Miscellaneous functions**  
(AFL 1.65)**SYNTAX**     *status( "statuscode" )***RETURNS**     ARRAY**FUNCTION**     Returns run–time status of the analysis engine. Supported status codes:

- "stocknum" – gives you the ordinal number of currently analysed symbol
- "action" – gives information in what context given formula is run: 1 – INDICATOR, 2 – COMMENTARY, 3 – SCAN, 4 – EXPLORATION, 5 – BACKTEST / OPTIMIZE
- "rangefromdate", "rangetodate" – return current auto–analysis From–To range as DateNums
- "rangefromtime", "rangetotime" – return current auto–analysis From–To range as DateNums
- "barinrange" – returns 1 when current bar is within current auto–analysis From–To range
- "barvisible" – (custom indicators only) returns 1 when current bar is visible in current view
- "firstbarinrange" and "lastbarinrange". They return 1 (or True) on the first/last bar of analysis range.
- "buydelay", "selldelay", "shortdelay", "coverdelay" – return delays set in the Settings window
- "firstbarintest" and "lastbarintest" – similar to "firstbarinrange" and "lastbarinrange" but they return the settings of last BACKTEST/OPTIMIZATION and intermediate scans/explorations do not affect them
- "firstvisiblebar", "lastvisiblebar", "firstvisiblebarindex", "lastvisiblebarindex" – return bar number or bar index of first/last visible bar. Available in indicator mode only.
- "redrawaction" – returns 0 (zero) for regular refreshes, and 1 for refreshes triggered via RequestTimedRefresh().
- "pxwidth" – returns pixel width of chart window pane (indicators only) (AmiBroker 4.94 or higher)
- "pxheight" – returns pixel height of chart window pane (indicators only) (AmiBroker 4.94 or higher)
- "axisminy" – retrieves the minimum (bottom) value of Y axis (indicators only)
- "axismaxy" – retrieves the maximum (top) value of Y axis (indicators only)

**EXAMPLE**     `if( Status( "redrawaction" ) ==1 )  
                  {  
                      _TRACE( "nTIMED REFRESH"+Now( ) ) ;  
                  }  
                  RequestTimedRefresh( 1 ) ;`

**SEE ALSO**     RequestTimedRefresh() function**References:**The **STATUS** function is used in the following formulas in AFL on–line library:

- [AFL Example – Enhanced](#)
- [Auto–Optimization Framework](#)
- [Calculate composites for tickers in list files](#)

- [Candle Identification](#)
- [Candle Stick Analysis](#)
- [Candle Stick Demo](#)
- [CCI\(20\) Divergence Indicator](#)
- [Color Display.afl](#)
- [Customised Avg. Profit %, Avg. Loss % etc](#)
- [DPO with shading](#)
- [Ed Seykota's TSP: EMA Crossover System](#)
- [Ed Seykota's TSP: Support and Resistance](#)
- [End Of Year Trading](#)
- [Gordon Rose](#)
- [MACD and histogram divergence detection](#)
- [MACD indicator display](#)
- [Performance Overview](#)
- [Pivot Finder](#)
- [prakash](#)
- [Ranking and sorting stocks](#)
- [Relative Strength Multichart of up to 10 tickers](#)
- [RUTVOL timing signal with BB Scoring routine](#)
- [Simple Candle Exploration](#)
- [TWS auto-export Executions-file parser](#)
- [Ultimate plus](#)
- [Using From and To dates from Auto Analysis in Code](#)

**More information:**

[Updated on-line reference](#)

**STDERR**  
**– standard error****Statistical functions**  
(AFL 1.4)**SYNTAX**     *StdErr( ARRAY, periods )***RETURNS**     ARRAY**FUNCTION**     Calculates standard error function (standard error of linear regression estimate) of the ARRAY over *periods* bars**EXAMPLE**     StdErr( close, 10 );**SEE ALSO****References:**

The **STDERR** function is used in the following formulas in AFL on–line library:

- [Standard Error Bands \(Native AFL\)](#)

**More information:**

[Updated on–line reference](#)

**STDEV****Statistical functions****– standard deviation**

(AFL 1.4)

**SYNTAX**      *StDev( ARRAY, periods )***RETURNS**    ARRAY**FUNCTION**    Calculates moving standard deviation of the ARRAY over *periods* bars**EXAMPLE**     stdev( close, 10 );**SEE ALSO****Comments:**

<b>Tomasz Janeczko</b>  2006-04-04 16:26:27	Note that if you are trying to compare results of StDev function to Excel output you should use STDEVP function in Excel (not StDev).
--	---

**References:**

The **STDEV** function is used in the following formulas in AFL on-line library:

- [% B of Bollinger Bands With Adaptive Zones](#)
- [Adaptive Price Channel](#)
- [AR\\_Prediction.afl](#)
- [BB squeeze](#)
- [Bollinger Band Width](#)
- [Bollinger oscillator](#)
- [CCT Bollinger Band Oscillator](#)
- [Cycle Highlighter](#)
- [Cycle Highlighter \(auto best-fit\)](#)
- [Dave Landry PullBack Scan](#)
- [DMI Spread Index](#)
- [Dynamic Momentum Index](#)
- [Dynamic Momentum Index](#)
- [Elder Triple Screen Trading System.](#)
- [Elder's SafeZone Stop](#)
- [Follow the Leader](#)
- [Historical Volatility Scan – 6/100](#)
- [Historical Volatility Scan – 50 Day](#)
- [Linear Regression Line w/ Std Deviation Channels](#)
- [Moving Trend Bands \(MTB\)](#)
- [MultiCycle 1.0](#)
- [nikhil](#)
- [NR4 Historical Volatility System](#)
- [Probability Calculator](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**

[Updated on-line reference](#)

**STOCHD****Indicators****– stochastic slow %D****SYNTAX**     *stochd( periods = 14, Ksmooth=3, Dsmooth=3 )***RETURNS**     ARRAY**FUNCTION**     Calculates the %D line of Stochastic Oscillator (with internal slowing KSmooth, DSmooth).**EXAMPLE**     The formula "stochd( 5 )" returns the value of a 5–period %D double smoothed by 3 periods**SEE ALSO**     [STOCHK\(\)](#) function**References:**

The **STOCHD** function is used in the following formulas in AFL on–line library:

- [Adaptave Zones O/B &O/S Oscillator](#)
- [Auto–Optimization Framework](#)
- [Bollinger Band Gap](#)
- [COMBO](#)
- [Dinapoli Guru Commentary](#)
- [Divergences](#)
- [Fund Screener](#)
- [hassan](#)
- [Stochastic Divergence, negative](#)
- [Stochastic Divergence, positive](#)
- [Stochastic Divergences, PDI, NDI](#)
- [Stochastic optimize](#)
- [Stochastic OSI &OBI](#)
- [Stochastics Trendlines](#)

**More information:**

[Updated on–line reference](#)

**STOCHK****Indicators****– stochastic slow %K****SYNTAX**     *stochk( periods = 14, ksmooth=3 )***RETURNS**     ARRAY**FUNCTION**     Calculates the %K line of Stochastic Oscillator (with internal slowing KSmooth).**EXAMPLE**     The formula "stochk( 5 )" returns the value of a 5–period %K slowed down 3 periods.**SEE ALSO**     [STOCHD\(\)](#) function**References:**

The **STOCHK** function is used in the following formulas in AFL on–line library:

- [Adaptave Zones O/B &O/S Oscillator](#)
- [Against all odds](#)
- [Auto–Optimization Framework](#)
- [CandleStochastics](#)
- [COMBO](#)
- [Dinapoli Guru Commentary](#)
- [hassan](#)
- [MACD and histogram divergence detection](#)
- [Stochastic OBV and Price Filter](#)
- [Stochastic optimize](#)
- [Stochastics Trendlines](#)

**More information:**

[Updated on–line reference](#)



**STREXTRACT****String manipulation****– extracts given item (substring) from comma-separated string**

(AFL 2.4)

**SYNTAX**     **StrExtract( list, item )****RETURNS**     STRING**FUNCTION**     Extracts given item (substring) from comma-separated list of items. item is a zero-based index of the item in the list.

If no substring at given index is found then empty string is returned ("").

Useful to retrieve symbols from the list obtained via GetCategorySymbols function.

**EXAMPLE**     StrExtract( "MSFT,AAPL,AMD,INTC", 2 ) will return AMD

StrExtract( "MSFT,AAPL,AMD,INTC", 0 ) will return MSFT

StrExtract( "MSFT,AAPL,AMD,INTC", 200 ) will return empty string ""

**SEE ALSO**     [GETCATEGORYSYMBOLS\(\)](#) function**References:**

The **StrExtract** function is used in the following formulas in AFL on-line library:

- [Baseline Relative Performance Watchlist charts V2](#)
- [Calculate composites for tickers in list files](#)
- [Candle Identification](#)
- [Count Tickers in Watchlist](#)
- [Ranking and sorting stocks](#)
- [Ranking Ticker WatchList](#)
- [Relative Strength](#)
- [TWS auto-export Executions-file parser](#)
- [WLBuildProcess](#)

**More information:**

[Updated on-line reference](#)

**STRFIND**  
– find substring in a string**String manipulation**  
(AFL 2.5)**SYNTAX**     *StrFind( string, substring )***RETURNS**     NUMBER**FUNCTION**     The **StrFind** function finds first occurrence of substring in string.

Returns 0 if not found, otherwise returns character index (one-based) of first occurrence.

**EXAMPLE**

```
if( StrFind( Name(), ".L" ) )
{
    printf( "The " + Name() + " has .L suffix " );
}
else
{
    printf( "The " + Name() + " does not have .L suffix " );
}
```

**SEE ALSO**     [StrExtract\(\)](#) function**References:**The **StrFind** function is used in the following formulas in AFL on–line library:

- [INTRADAY HEIKIN ASHI new](#)
- [Pivots for Intraday Forex Charts](#)

**More information:**[Updated on–line reference](#)

**STRFORMAT****String manipulation****– Write formatted output to the string**

(AFL 2.5)

**SYNTAX**     **StrFormat( formatstr, ... )****RETURNS**     STRING**FUNCTION**     The **StrFormat** function formats and returns a series of characters and values in the result string.

If arguments follow the format string, the format string must contain specifications that determine the output format for the arguments.

StrFormat and printf behave identically except that printf writes output to the window, while StrFormat does not write anything to output window but returns resulting string instead.

StrFormat function is useful with conjunction with **fputs** function that allows to write string to a file.

Note 1: for numbers always use %f, %e or %g formatting, %d or %x will not work because there are no integers in AFL.

Note 2: as of now only numbers and arrays can now be used. For arrays 'selected value' is used

**EXAMPLE**

```
fh = fopen( "Test.csv", "w" );
for( i = 0; fh &i < 10; i++ )
{
    text = StrFormat( "Hello world, line %g\n", i );
    fputs( text, fh );
}

fclose( fh );
```

**SEE ALSO**     printf() function , fputs() function**References:**

The **StrFormat** function is used in the following formulas in AFL on–line library:

- [Andrews PitchforkV3.3](#)
- [DateNum\\_DateStr](#)
- [Dave Landry PullBack Scan](#)
- [Elder safe Zone Long + short](#)
- [Elder Triple Screen Trading System.](#)
- [Export Intraday Data](#)
- [FastStochK FullStochK–D](#)
- [Fibonacci Moving averages](#)
- [LunarPhase](#)
- [nifty](#)
- [Option Calls, Puts and days till third friday.](#)
- [prakash](#)
- [SAR–ForNextBarStop](#)

- [Schiff Lines](#)

**More information:**

[Updated on-line reference](#)

**STRLEFT****String manipulation****– extracts the leftmost part**

(AFL 2.0)

**SYNTAX**      *strleft( STRING, count)***RETURNS**     STRING**FUNCTION**     Extracts the first (that is, leftmost) *count* characters from STRING and returns a copy of the extracted substring. If *count* exceeds the string length, then the entire string is extracted.**EXAMPLE**      newstring = strleft( string, 4 );**SEE ALSO****References:**

The **StrLeft** function is used in the following formulas in AFL on–line library:

- [Auto–Optimization Framework](#)
- [DateNum\\_DateStr](#)
- [Date\\_To\\_Num\(\), Time\\_To\\_Num\(\)](#)
- [Futures – Dollar Move Indicator](#)
- [Futures – Dollar Move Today Indicator](#)
- [Gordon Rose](#)
- [Pivot Finder](#)
- [Ranking and sorting stocks](#)
- [Time Left in Bar](#)
- [TWS auto–export Executions–file parser](#)

**More information:**

[Updated on–line reference](#)

**STRLEN**  
– string length**String manipulation**  
(AFL 1.5)**SYNTAX**     *strlen( STRING )***RETURNS**    NUMBER**FUNCTION**   calculates the length of the string**EXAMPLE**    This function could be used for (for example) filtering out only 3 letter stock codes: buy = something AND **strlen**( name() ) == 3;**SEE ALSO****References:**

The **StrLen** function is used in the following formulas in AFL on–line library:

- [Calculate composites for tickers in list files](#)
- [Ranking and sorting stocks](#)
- [Time Left in Bar](#)
- [TWS auto–export Executions–file parser](#)

**More information:**

[Updated on–line reference](#)

**STRMID****String manipulation****– extracts part of the string**

(AFL 2.0)

**SYNTAX**     *StrMid( STRING, start, count)***RETURNS**     STRING**FUNCTION**     Extracts a substring of length *count* characters from STRING, starting at position *start* (zero-based). The function returns a copy of the extracted substring.**EXAMPLE**     newstring = strmid( string, 1, 2 );**SEE ALSO****References:**

The **StrMid** function is used in the following formulas in AFL on-line library:

- [DateNum\\_DateStr](#)
- [Date\\_To\\_Num\(\)](#), [Time\\_To\\_Num\(\)](#)
- [Futures – Dollar Move Indicator](#)
- [Futures – Dollar Move Today Indicator](#)
- [TWS auto-export Executions-file parser](#)

**More information:**

[Updated on-line reference](#)

## STRREPLACE

### – string replace

String manipulation  
(AFL 2.90)

**SYNTAX**     *StrReplace( srcstring, oldsubstring, newsubstring )*

**RETURNS**    STRING

**FUNCTION**    This function returns a string with all occurrences of *oldsubstring* in *srcstring* replaced with the given *newsubstring* value. The string may grow or shrink as a result of the replacement, that is *oldsubstring* and *newsubstring* do not have to be equal in length. The function performs case-sensitive matches.

**EXAMPLE**     *// the expression below will*  
                  *// result in string in which 'red' is replaced with 'brown'*  
                  *StrReplace("This fox is red", "red", "brown" );*

**SEE ALSO**     [StrExtract\(\)](#) function , [StrFind\(\)](#) function , [StrFormat\(\)](#) function , [StrLeft\(\)](#) function , [StrLen\(\)](#) function , [StrMid\(\)](#) function , [StrRight\(\)](#) function , [StrToDateTime\(\)](#) function , [StrToLower\(\)](#) function , [StrToNum\(\)](#) function , [StrToUpper\(\)](#) function

#### References:

The **StrReplace** function is used in the following formulas in AFL on-line library:

- [Calculate composites for tickers in list files](#)
- [WLBuildProcess](#)

#### More information:

[Updated on-line reference](#)



**STRRIGHT****String manipulation****– extracts the rightmost part of the string**

(AFL 2.0)

**SYNTAX**     *StrRight( STRING, count)***RETURNS**     STRING**FUNCTION**     Extracts the last (that is, rightmost) *count* characters from STRING and returns a copy of the extracted substring. If *count* exceeds the string length, then the entire string is extracted.**EXAMPLE**     newstring = stright( string, 4 );**SEE ALSO****References:**

The **StrRight** function is used in the following formulas in AFL on–line library:

- [DateNum\\_DateStr](#)
- [Date\\_To\\_Num\(\)](#), [Time\\_To\\_Num\(\)](#)
- [Relative Strength Multichart of up to 10 tickers](#)
- [Time Left in Bar](#)
- [TWS auto–export Executions–file parser](#)

**More information:**[Updated on–line reference](#)

**STRTODATETIME**  
**– convert string to datetime****String manipulation**  
(AFL 2.80)**SYNTAX**     **StrToDateTime( "string" )****RETURNS**     NUMBER**FUNCTION**     Converts string representing date/time value to the corresponding DateTime number (that can be later compared to output of DateTime() function for example).**EXAMPLE**     `Buy = DateTime( ) == StrToDateTime( "2005-Mar-05" );`**SEE ALSO**     [DATETIME\(\)](#) function , [DateTimeToStr\(\)](#) function**References:**

The **StrToDateTime** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

## STRTOLOWER

– convert to lowercase

String manipulation  
(AFL 2.80)

**SYNTAX**     *StrToLower( "string" )*

**RETURNS**    STRING

**FUNCTION**    This function converts input string to all lower case.

**EXAMPLE**     `Title = StrToLower( "MiXeD CaSe" );`

**SEE ALSO**     [StrToUpper\(\)](#) function

### References:

The **StrToLower** function is used in the following formulas in AFL on–line library:

### More information:

[Updated on–line reference](#)

## STRTONUM

– convert string to number

String manipulation  
(AFL 2.5)

**SYNTAX**     *StrToNum( string )*

**RETURNS**    NUMBER

**FUNCTION**    Converts string to number.

**EXAMPLE**     List = "123,456,789";

```
for( i = 0; ( Item = StrExtract( List, i ) ) != ""; i++ )
{
    printf( "%gn", StrToNum( Item ) );
}
```

**SEE ALSO**     [WRITEVAL\(\)](#) function

### References:

The **StrToNum** function is used in the following formulas in AFL on–line library:

- [DateNum\\_DateStr](#)
- [Date\\_To\\_Num\(\)](#), [Time\\_To\\_Num\(\)](#)
- [TWS auto–export Executions–file parser](#)

### More information:

[Updated on–line reference](#)

**STRTOUPPER****String manipulation****– convert to uppercase**

(AFL 2.80)

**SYNTAX**     *StrToUpper( "string" )***RETURNS**     STRING**FUNCTION**     This function converts input string to all upper case.**EXAMPLE**     `Title = StrToUpper( "MiXeD CaSe" );`**SEE ALSO**     [StrToLower\(\)](#) function**References:**

The **StrToUpper** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

## STUDY

### – reference hand–drawn study

Miscellaneous functions  
(AFL 1.5)

**SYNTAX**     *study*( *STUDYID*, *CHARTID* = 1 )

**RETURNS**     ARRAY

**FUNCTION**     generates an array equivalent to a trendline study drawn by the user – allows detecting trendline breakouts from AFL.  
**STUDYID** is a two–character identifier of the study. identifiers are: "UP" – uptrend, "DN" – downtrend, "SU" – support, "RE" – resistance, "ST" – stop loss, however you can use ANY identifiers (there are no limitations except that AmiBroker accepts only 2 letter codes).  
**CHARTID** – identifies the chart pane where the study was drawn – you can find out what is the chart ID for given chart by looking in Parameters dialog, Axes &Grid, Miscellaneous: Chart ID or using GetChartID() AFL function.

More information about this function is included in the [Tutorial: Using Studies in AFL formulas](#)

**EXAMPLE**     *// this example plots filled area between*  
*// support (SU) and resistance (RE) lines*

```
Plot(C, "Price", colorBlack, styleCandle );
su = Study( "SU", GetChartID() );
re = Study( "RE", GetChartID() );
PlotOHLC( re, re, su, su, "", colorYellow, styleCloud );
```

**SEE ALSO**     [GETCHARTID\(\)](#) function

#### References:

The **STUDY** function is used in the following formulas in AFL on–line library:

#### More information:

[Updated on–line reference](#)

**SUM**

Moving averages, summation

**– sum data over specified number of bars****SYNTAX**     *sum( ARRAY, periods )***RETURNS**     ARRAY**FUNCTION**     Calculates a cumulative sum of the ARRAY for the specified number of lookback *periods* (including today).**EXAMPLE**     The formula "sum( CLOSE, 14 )" returns the sum of the preceding 14 closing prices. A 14–period simple moving average could be written "sum(C,14) / 14."**SEE ALSO**     [CUM\(\)](#) function**Comments:**

<p><b>Graham Kavanagh</b>  gkavanagh@e-wire.net.au  2004–08–09 07:52:41</p>	<p>Sum adds up the last "n" number of bars. It sums whatever you put into the first part of the sum formula.</p> <p>Cum(1) adds 1 to the previous value of Cum, so the first bar is 1 and it just keeps adding one to the last bar value of cum(1).  You can use Cum to add anything, like how many times you get rising days in the entire chart:</p> <p>Rise = C&gt;O; //this gives results of 0 or 1  TotalRise = Cum(Rise);</p> <p>You could limit this as well to time periods, or any other condition  Example would be one for total rise days since 1995:</p> <p>RecentRise = C&gt;O and Year()&gt;=1995; //this gives results of 0 or 1  TotalRise = Cum(RecentRise);</p> <p>If you wanted to know how many rising days in the last 12 bars you would use:</p> <p>LastRises = Sum(Rise,12);</p> <p>Hope this helps</p>
---	---

**References:**

The **SUM** function is used in the following formulas in AFL on–line library:

- [Against all odds](#)
- [AJDX system](#)
- [Alpha and Beta and R\\_Squared Indicator](#)
- [AR\\_Prediction.afl](#)
- [Auto–Optimization Framework](#)
- [Bollinger band normalization](#)
- [Buff Volume Weighted Moving Averages](#)

- CandleStick Comentary---Help needed
- CandleStochastics
- Chaikin Money Flow
- Chande Momentum Oscillator
- Cole
- crBeta
- crMathLib
- DeMarker
- Dynamic Momentum Index
- Dynamic Momentum Index
- ekeko price chart
- Elder safe Zone Long + short
- Elder Triple Screen Trading System.
- Elder's SafeZone Stop
- ElderSafeZoneStopLong
- ElderSafeZoneStopShort
- Fib CMO
- Hilbert Study
- Hurst Constant
- Linear Regression Line & Bands
- MACD commentary
- Market Profile & Market Volume Profile
- MultiCycle 1.0
- Performance Check
- Peterson
- Polarized Fractal Efficiency
- Projection Oscillator
- QP2 Float Analysis
- Range Expansion Index
- Regression Analysis Line
- Relative Vigour Index
- RSI of volume weighted moving average
- RSI of Weekly Price Array
- RSIS
- Sector Tracking
- Stochastic of Weekly Price Array
- TD sequential
- The Saturation Indicator D\_sat
- Time Frame Weekly Bars
- Time segment value
- Trend Analysis\_Comentary
- Trend Continuation Factor
- Trigonometric Fit – TrigFit with AR for cos / sin
- Vertical Horizontal Filter
- Vertical Horizontal Filter (VHF)
- Visualization of stoploses and profit in chart
- Volume Weighted Moving Average

#### More information:

Updated on-line reference





**TAN****Math functions****– tangent function**

(AFL 1.0)

**SYNTAX**      *tan( NUMBER )*  
                 *tan(ARRAY)*

**RETURNS**    NUMBER,ARRAY

**FUNCTION**   Returns the tangent of NUMBER or ARRAY. This function assumes that the ARRAY values are in radians

**EXAMPLE**

**SEE ALSO**    [atan\(\)](#) function

**References:**

The **tan** function is used in the following formulas in AFL on–line library:

- [Andrews Pitchfork](#)
- [Andrews PitchforkV3.3](#)
- [AR\\_Prediction.afl](#)
- [DMI Spread Index](#)
- [Hilbert Study](#)
- [Moving Average "Crash" Test](#)
- [Multiple sinus noised](#)
- [Schiff Lines](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)

**More information:**

[Updated on–line reference](#)

## TANH – hyperbolic tangent function

**Math functions**  
(AFL 2.80)

**SYNTAX**     *tanh( NUMBER )*  
                 *tanh( ARRAY )*

**RETURNS**    NUMBER,  
                 ARRAY

**FUNCTION**   Returns the hyperbolic tangent of NUMBER or ARRAY. This function assumes that the ARRAY values are in radians.

### EXAMPLE

### SEE ALSO

#### References:

The **tanh** function is used in the following formulas in AFL on–line library:

#### More information:

[Updated on–line reference](#)

## TEMA – triple exponential moving average

Moving averages, summation  
(AFL 2.0)

**SYNTAX**     *tema( ARRAY, periods )*

**RETURNS**     ARRAY

**FUNCTION**     Calculates triple exponentially smoothed average – TEMA. The function accepts time–variable *periods*.

**EXAMPLE**     TEMA( Close, 5 )

**SEE ALSO**     MA(), EMA(), WMA(), DEMA()

### References:

The **TEMA** function is used in the following formulas in AFL on–line library:

- [Auto–Optimization Framework](#)
- [Balance of Power](#)
- [balance of power](#)
- [BMTRIX Intermediate Term Market Trend Indicator](#)
- [Bull/Bear Volume](#)
- [Dahl Oscillator modified](#)
- [Elder Impulse Indicator V2](#)

### More information:

[Updated on–line reference](#)

**TIMEFRAMECOMPRESS**Time Frame functions  
(AFL 2.5)**– compress single array to given time frame****SYNTAX**     *TimeFrameCompress( array, interval, mode = compressLast )***RETURNS**     ARRAY**FUNCTION**     The **TimeFrameCompress** function compresses single array to given interval using given compression mode available modes:

- compressLast – last (close) value of the array within interval
- compressOpen – open value of the array within interval
- compressHigh – highest value of the array within interval
- compressLow – lowest value of the array within interval
- compressVolume – sum of values of the array within interval

To expand compressed array you should use the **TimeFrameExpand** function.

The **TimeFrameCompress** function is provided for completeness and it can be used when you want to compress single array without affecting built-in OHLC,V arrays. If you call TimeFrameCompress it does not affect results of other functions (opposite to **TimeFrameSet**).

For more information check [Tutorial: Multiple time frame support](#)

**EXAMPLE**

```

wc = TimeFrameCompress( Close, inWeekly );

/* now the time frame is still unchanged (say daily) and our MA will
operate on daily data */
dailyma = MA( C, 14 );

/* but if we call MA on compressed array, it will give MA from other
time frame */
weeklyma = MA( wc, 14 ); // note that argument is time-compressed
array

Plot( dailyma, "DailyMA", colorRed );

weeklyma = TimeFrameExpand( weeklyma, inWeekly ); // expand for
display

Plot( weeklyma, "WeeklyMA", colorBlue );

```

**SEE ALSO**     [TimeFrameExpand\(\)](#) function

**References:**

The **TimeFrameCompress** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)



**TIMEFRAMEEXPAND****Time Frame functions****– expand time frame compressed array**

(AFL 2.5)

**SYNTAX**     *TimeFrameExpand( array, interval, mode = expandLast )***RETURNS**    ARRAY**FUNCTION**    The **TimeFrameExpand** function expands time-compressed *array* from *interval* time frame to base time frame (*interval* parameter must match the value used in **TimeFrameCompress** or **TimeFrameSet**)

The **TimeFrameExpand** is used to decompress array variables that were created in different time frame. Decompressing is required to properly display the array created in different time frame. For example if you want to display weekly moving average it must be 'expanded' so the data of one weekly bar covers five daily bars (Monday–Friday) of corresponding week.

Available *modes*:

- **expandLast** – the compressed value is expanded starting from last bar within given period (so for example weekly close/high/low is available on Friday's bar)
- **expandFirst** – the compressed value is expanded starting from first bar within given period (so for example weekly open is available from Monday's bar)
- **expandPoint** – the resulting array gets not empty values only for the last bar within given period (all remaining bars are Null (empty)).

Caveat: **expandFirst** used on price different than open may look into the future. For example if you create weekly HIGH series, expanding it to daily interval using **expandFirst** will enable you to know on MONDAY what was the high for entire week.

For more information check [Tutorial: Multiple time frame support](#)

**EXAMPLE**

```

wc = TimeFrameCompress( Close, inWeekly );

/* now the time frame is still unchanged (say daily) and our MA will
operate on daily data */
dailyma = MA( C, 14 );

/* but if we call MA on compressed array, it will give MA from other
time frame */
weeklyma = MA( wc, 14 ); // note that argument is time-compressed
array

Plot( dailyma, "DailyMA", colorRed );

weeklyma = TimeFrameExpand( weeklyma, inWeekly ); // expand for
display

Plot( weeklyma, "WeeklyMA", colorBlue );

```

**SEE ALSO**    [TimeFrameSet\(\)](#) function , [TimeFrameRestore\(\)](#) function

**References:**

The **TimeFrameExpand** function is used in the following formulas in AFL on–line library:

- [EKEKO SAR–MF](#)
- [Elder Impulse Indicator](#)
- [Elder Impulse Indicator V2](#)
- [IFT of RSI – Multiple TimeFrames](#)
- [Pivot Point with S/R Trendlines](#)
- [Rea Time Daily Price Levels](#)

**More information:**

[Updated on–line reference](#)



**TIMEFRAMEGETPRICE**

Time Frame functions

– retrieve O, H, L, C, V values from other time frame

(AFL 2.5)

**SYNTAX**     *TimeFrameGetPrice( pricefield, interval, shift = 0, mode = expandFirst )***RETURNS**     ARRAY**FUNCTION**     The **TimeFrameGetPrice** – retrieves OHLCV fields from other time frames. This works immediately without need to call **TimeFrameSet** at all.

First parameter – *pricefield* – is one of the following: "O", "H", "L", "C", "V", "I" (open interest).

*Interval* is bar interval in seconds. You can use pre-defined interval constants: in1Minute, in5Minute, in15Minute, inHourly, inDaily, inWeekly, inMonthly. Or integer multiples like (3\*in1Minute) for 3 minute bars

*shift* allows to reference past (negative values) and future (positive values) data in higher time frame. For example –1 gives previous bar's data (like in Ref function but this works in higher time frame).

*mode* – one of available modes:

- expandLast – the compressed value is expanded starting from last bar within given period (so for example weekly close/high/low is available on Friday's bar)
- expandFirst – the compressed value is expanded starting from first bar within given period (so for example weekly open is available from Monday's bar)
- expandPoint – the resulting array gets not empty values only for the last bar within given period (all remaining bars are Null (empty)).

Note these functions work like these 3 nested functions:

```
TimeFrameExpand( Ref( TimeFrameCompress( array, interval,
compress(depending on field used) ), shift ), interval, expandFirst
)
```

therefore, if shift = 0 compressed data may look into the future ( weekly high can be known on monday ). If you want to write a trading system using this function please make sure to reference PAST data by using negative shift value.

The only difference is that TimeFrameGetPrice is 2x faster than nested Expand/Compress.

For more information check [Tutorial: Multiple time frame support](#)

**EXAMPLE**

```
// Example 1. get previous week Open price
TimeFrameGetPrice( "O", inWeekly, -1 )

// Example 2. get weekly Close price 3 weeks ago
TimeFrameGetPrice( "C", inWeekly, -3 )

// Example 3. get weekly High price 2 weeks ago
TimeFrameGetPrice( "H", inWeekly, -2 )
```

```
// Example 4. get this week Open price.  
TimeFrameGetPrice( "O", inWeekly, 0 )
```

```
// Example 5. get previous Day High when working on intraday data  
TimeFrameGetPrice( "H", inDaily, -1 )
```

**SEE ALSO** [TimeFrameSet\(\)](#) function

#### References:

The **TimeFrameGetPrice** function is used in the following formulas in AFL on–line library:

- [Market Profile &Market Volume Profile](#)
- [Pivots for Intraday Forex Charts](#)

#### More information:

[Updated on–line reference](#)

**TIMEFRAMEMODE****Time Frame functions****– switch time frame compression mode**

(AFL 2.80)

**SYNTAX**     *TimeFrameMode( mode )***RETURNS**    NOTHING**FUNCTION**   Switches time frame functions to different operating modes. Where mode is one of 0, 1, or 2.

- TimeFrameMode( 0 );  
– switches time frame functions to time–based operation (the default)
- TimeFrameMode( 1 );  
– switches time frame functions to N–tick operation (positive values passed to TimeFrameSet are treated now as N–tick)
- TimeFrameMode( 2 );  
– switches time frame functions to N–volume bar operation (positive values passed to TimeFrameSet are treated now as N–volume bars)
- TimeFrameMode( 3 );  
– switches time frame functions to N–Range bar operation (positive values passed to TimeFrameSet are treated now as N–range bars)

Note: N–volume bars are very different from time–based bars (compression of data to N–volume bar may actually deliver MORE output bars – for example if one tick is 1000 shares and you have specified 100V bars then single tick will be expanded to TEN 100V bars – ten times original size)

TimeFrame functions are protected against array overrun and will not decompress beyond original array size (you will get an "Error 47. N–volume bar compressed data longer than base time frame"). Also switching main time frame to some weird N–volume bar value will result in limiting the output to maximum twice original data size (without error message).

You should keep that in mind and avoid using too small N–volume bar intervals that could lead to such condition. Also due to the nature of N–volume bars the only TimeFrameSet() function will yield correct N–volume bar values, TimeFrameGetPrice() may give slightly distorted results.

It is also possible to use n–volume bars in TimeFrame functions without calling TimeFrameMode() – it is then necessary to specify n–volume bars as negative number offset by –1000000 (minus one million):

```
TimeFrameSet( –1000000 – 2000 );
```

**EXAMPLE**

```
TimeFrameMode( 2 );
TimeFrameSet( 50000 ); // 50'000 share bars..
//...do something ...
TimeFrameRestore();
```

**SEE ALSO**    TimeFrameSet() function , TimeFrameRestore() function

**References:**

The **TimeFrameMode** function is used in the following formulas in AFL on-line library:

**More information:**

[Updated on-line reference](#)

**TIMEFRAMERESTORE****Time Frame functions****– restores price arrays to original time frame**

(AFL 2.5)

**SYNTAX**     *TimeFrameRestore()***RETURNS**    NOTHING**FUNCTION**    The **TimeFrameRestore** function restores price arrays replaced by **TimeFrameSet**.

Note that only OHLC, V, OI and Avg built-in variables are restored to original time frame when you call **TimeFrameRestore()**.

All other variables created when being in different time frame remain compressed.

To de-compress them to original interval you have to use **TimeFrameExpand**.

**EXAMPLE**

```

TimeFrameSet( in5Minute ); // switch to 5 minute frame

/* MA now operates on 5 minute data, ma5_13 holds time-compressed 13
bar MA of 5min bars */

ma5_13 = MA( C, 13 );

TimeFrameRestore(); // restore time frame to original

TimeFrameSet( inHourly ); // switch now to hourly

mah_9 = EMA( C, 9 ); // 9 bar moving average from hourly data

TimeFrameRestore(); // restore time frame to original

Plot( Close, "Price", colorWhite, styleCandle );

// plot expanded average

Plot( TimeFrameExpand( ma5_13, in5Minute ), "13 bar moving average
from 5 min bars", colorRed );
Plot( TimeFrameExpand( mah_9, inHourly ), "9 bar moving average from
hourly bars", colorRed );

```

**SEE ALSO**    **TimeFrameSet()** function**Comments:**

<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2004-07-10 06:19:47	TimeFrameRestore and RestorePriceArrays is essentially the same function. So please note that calling TimeFrameRestore also resets the ticker set by eventual previous call to SetForeign()
--	---

**References:**

The **TimeFrameRestore** function is used in the following formulas in AFL on–line library:

- [EKEKO SAR–MF](#)
- [Elder Impulse Indicator](#)
- [Elder Impulse Indicator V2](#)
- [Elder Triple Screen Trading System.](#)
- [IFT of RSI – Multiple TimeFrames](#)
- [Lagging MA–Xover](#)
- [Pivot Point with S/R Trendlines](#)
- [Price with Woodies Pivots](#)
- [Rea Time Daily Price Levels](#)
- [SUPER PIVOT POINTS](#)

**More information:**

[Updated on–line reference](#)

**TIMEFRAMESET**Time Frame functions  
(AFL 2.5)**– switch price arrays to a different time frame****SYNTAX**     *TimeFrameSet( interval)***RETURNS**    NOTHING

**FUNCTION**    The **TimeFrameSet** replaces current price/volume arrays: open, high, low, close, volume, openint, avg with time-compressed bars of specified interval once you switched to a different time frame all calculations and built-in indicators operate on selected time frame. To get back to original interval call **TimeFrameRestore()** function. Before calling **TimeFrameSet** again in the same formula with different interval you have to restore original time frame first using **TimeFrameRestore**.

*interval* defines time frame interval in seconds. So 60 means 1-minute. For the convenience the following interval constants are pre-defined:

- in1Minute = 60
- in5Minute = 5 \* 60
- in15Minute = 15 \* 60
- inHourly = 3600
- inDaily = 24 \* 3600
- inWeekly = 5 \* 24 \* 3600 + 1 = 432001
- inMonthly = 25 \* 24 \* 3600 + 1 = 2160001

To get other intervals you can use multiple of pre-defined intervals, for example: ( 3\*in1Minute ) gives 3 minute bars. Or you can use 3 \* inDaily for 3-day bars.

New in version 4.70 and above: You can also use NEGATIVE values for N-tick charts: –5 gives 5-tick chart. Note that N-tick compression works correct only if you have 1-tick base time interval selected in database settings.

You can also use TimeFrameSet to create N-volume bars as well as Range bars. See [TimeFrameMode\(\)](#) function for more details.

**VERY IMPORTANT:**

inWeekly constant is now 432001 ( 5\*inDaily + 1 ) – in previous version it was 432000  
 inMonthly constant is now 2160001 ( 25\*inDaily + 1 ) – in previous version it was 2160000  
 It is changed because now N-day custom intervals are supported and they will interfere with weekly/monthly.  
 Note that 5\*inDaily is now DIFFERENT than inWeekly. 5\*inDaily creates 5-day bars that DO NOT necessarily cover Monday–Friday while inWeekly ALWAYS creates bars that begin on Monday and end on Friday. Also 25\*inDaily creates 25-day bars that DO NOT necessarily represent full month, while inMonthly always begins with first day of the month and ends at the last day of the month

Once you switch the time frame using **TimeFrameSet** , all AFL functions operate on this time frame until you switch back the time frame to original interval using **TimeFrameRestore** or set to different interval again using **TimeFrameSet**. It is good idea to ALWAYS call **TimeFrameRestore** when you are done with processing in other time frames.

When time frame is switched to other than original interval the results of all functions called since **TimeFrameSet** are time-compressed too. If you want to display them in original time frame you would need to 'expand' them as described later. Variables created and assigned before call to **TimeFrameSet()** remain in the time frame they were created. This behaviour allows mixing unlimited different time frames in single formula.

Please note that you can only compress data from shorter interval to longer interval. So when working with 1-minute data you can compress to 2, 3, 4, 5, 6, ....N-minute data. But when working with 15 minute data you can not get 1-minute data bars. In a similar way if you have only EOD data you can not access intraday time frames.

For more information check: [Tutorial: Multiple time frame support in AFL](#)

**EXAMPLE**

```
TimeFrameSet( in5Minute ); // switch to 5 minute frame

/* MA now operates on 5 minute data, ma5_13 holds time-compressed 13
bar MA of 5min bars */

ma5_13 = MA( C, 13 );

TimeFrameRestore(); // restore time frame to original

TimeFrameSet( inHourly ); // switch now to hourly

mah_9 = EMA( C, 9 ); // 9 bar moving average from hourly data

TimeFrameRestore(); // restore time frame to original

Plot( Close, "Price", colorWhite, styleCandle );

// plot expanded average

Plot( TimeFrameExpand( ma5_13, in5Minute), "13 bar moving average
from 5 min bars", colorRed );
Plot( TimeFrameExpand( mah_9, inHourly), "9 bar moving average from
hourly bars", colorRed );
```

**SEE ALSO** [TimeFrameRestore\(\)](#) function , [TimeFrameExpand\(\)](#) function , [TimeFrameGetPrice\(\)](#) function

**Comments:**

<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2004-06-03 04:35:37	TimeFrameSet(in15Minute); MA10_15Min=MA(Close,10); TimeFrameRestore();  Buy=Cross( MA(Close,5), TimeFrameExpand(MA10_15Min, in15Minute) );
--	--

**References:**

The **TimeFrameSet** function is used in the following formulas in AFL on-line library:

- [EKEKO SAR-MF](#)
- [Elder Impulse Indicator](#)



- [Elder Impulse Indicator V2](#)
- [Elder Triple Screen Trading System.](#)
- [IFT of RSI – Multiple TimeFrames](#)
- [Lagging MA–Xover](#)
- [Pivot Point with S/R Trendlines](#)
- [Price with Woodies Pivots](#)
- [Rea Time Daily Price Levels](#)
- [SUPER PIVOT POINTS](#)

**More information:**

[Updated on–line reference](#)

**TIMENUM****Date/Time**  
(AFL 2.0)**– get current bar time****SYNTAX**     *timenum()***RETURNS**    ARRAY**FUNCTION**    Returns the array with numbers that represent quotation time coded as follows:  
10000 \* hour + 100 \* minute + second, so 12:37:15 becomes 123715**EXAMPLE**     TimeNum()**SEE ALSO**     Hour(), Minute(), Second(), TimeNum()**References:**

The **TIMENUM** function is used in the following formulas in AFL on–line library:

- [AR\\_Prediction.afl](#)
- [Date\\_To\\_Num\(\), Time\\_To\\_Num\(\)](#)
- [Lagging MA–Xover](#)
- [Moving Averages NoX](#)
- [Rea Time Daily Price Levels](#)
- [Time Left in Bar](#)
- [Trigonometric Fit – TrigFit with AR for cos / sin](#)
- [TWS auto–export Executions–file parser](#)

**More information:**[Updated on–line reference](#)

## TRIN

### – traders (Arms) index

**Composites**  
(AFL 1.2)

**SYNTAX**     *trin()*

**RETURNS**    ARRAY

**FUNCTION**    Calculates TRIN (Arms Index) indicator.

NOTE: All built-in a/d indicators (AdLine/Trin) work only with composites calculated inside AmiBroker <http://www.amibroker.com/newsletter/04-2000.html>

If you are using QP2 database for example you should use QP2's own symbols for advances/declines.

!NY-A, !NY-D, !NY-AV, !NY-DV

The formula for NYSE TRIN using QP2 database is:

```
ArmsIndex = ( Foreign("!NY-A", "C") / Foreign("!NY-D", "C") ) / (
Foreign("!NY-AV", "C") / Foreign("!NY-DV", "C") );
Plot( ArmsIndex, "TRIN", colorRed );
```

**EXAMPLE**     *trin()*

#### SEE ALSO

#### References:

The **TRIN** function is used in the following formulas in AFL on-line library:

#### More information:

[Updated on-line reference](#)

**TRIX****Indicators****– triple exponential smoothed price****SYNTAX**     *trix( periods = 9 )***RETURNS**     ARRAY**FUNCTION**     Calculates the TRIX indicator (with averaging range of *periods*).**EXAMPLE**     *trix( 12 )***SEE ALSO****References:**

The **TRIX** function is used in the following formulas in AFL on–line library:

- [TRIX](#)
- [TRIXXX](#)

**More information:**

[Updated on–line reference](#)

## TROUGH

### – trough

**Basic price pattern detection**  
(AFL 1.1)

**SYNTAX**      *trough*(ARRAY, change, n = 1)

**RETURNS**    ARRAY

**FUNCTION**    Gives the value of ARRAY *n*-th trough(s) ago. This uses the Zig Zag function (see Zig Zag) to determine the troughs. **Caveat:** this function is based on Zig-Zag indicator and may look into the future.

**EXAMPLE**    *trough*(close,5,1)

### SEE ALSO

#### Comments:

<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2003-02-13 04:01:33	Zig/Peak/Trough functions work correctly for ARRAYS containing data greater than zero.
---	--

#### References:

The **TROUGH** function is used in the following formulas in AFL on-line library:

- [Andrews Pitchfork](#)
- [Andrews PitchforkV3.3](#)
- [Automatic Trend-line](#)
- [Constant Trendline Plot](#)
- [Gartley 222 Pattern Indicator](#)
- [MACD commentary](#)
- [Pivot Point with S/R Trendlines](#)
- [Schiff Lines](#)
- [Support Resistance levels](#)
- [Tom DeMark Trend Lines](#)

#### More information:

[Updated on-line reference](#)

**TROUGHBARS****– bars since trough****Basic price pattern detection**  
(AFL 1.1)**SYNTAX**     *troughbars*(ARRAY, change, n = 1)**RETURNS**     ARRAY**FUNCTION**     Plots the number of bars that have passed from the *n*-th trough. This uses the Zig Zag function (see Zig Zag) to determine the troughs. **Caveat:** this function is based on Zig-Zag indicator and may look into the future.**EXAMPLE**     *troughbars*(close,5,1)**SEE ALSO****References:**

The **TROUGHBARS** function is used in the following formulas in AFL on-line library:

- [Constant Trendline Plot](#)
- [Gartley 222 Pattern Indicator](#)
- [Head & Shoulders Pattern](#)
- [Pattern Recognition Exploration](#)
- [Pivot Point with S/R Trendlines](#)
- [QP2 Float Analysis](#)
- [RSI Trendlines and Wedges](#)
- [Stochastics Trendlines](#)
- [The Fibonacci behavior](#)
- [Tom DeMark Trend Lines](#)

**More information:**

[Updated on-line reference](#)

**TSF****– time series forecast****Statistical functions**  
(AFL 2.2)**SYNTAX**     ***TSF***(***ARRAY***, ***periods***)**RETURNS**     ***ARRAY*****FUNCTION**     Calculates time series forecast indicator (similar to LinearReg but differs by the value of lin reg slope)**EXAMPLE**     Plot( Close, "Price", colorBlue, styleCandle );  
Plot( TSF(close,5), "Time Series Forecast", colorRed );**SEE ALSO****Comments:**

<b>Nigel Rowe</b>  2003-04-30 06:03:00	TSF is exactly the estimate of LinearReg for the NEXT DAY.  (it is calculated as LinearReg PLUS LinRegSlope * 1 (bar))  Plot(LinearReg(Close, 10 )+LinRegSlope(Close, 10), "Forecast for tommorrow", colorRed );  Plot(TSF(Close, 10 ), "Forecast for tommorrow 2", colorBlue );
---	--

**References:**

The **TSF** function is used in the following formulas in AFL on-line library:

- [Moving Trend Bands \(MTB\)](#)

**More information:**

[Updated on-line reference](#)

**ULTIMATE****Indicators****– ultimate oscillator****SYNTAX**      *ultimate( fast = 7, med = 14, slow = 28 )***RETURNS**    ARRAY

**FUNCTION**    Calculates the Ultimate Oscillator indicator using the three cycle lengths supplied as parameters. Note that each of the three parameters must be greater than the preceding parameter.

**EXAMPLE**     The formula "ultimate( 7, 14, 21 )" returns the default Ultimate Oscillator.

**SEE ALSO****References:**

The **ULTIMATE** function is used in the following formulas in AFL on–line library:

- [Adaptive Zones O/B &O/S Oscillator](#)
- [RSI "based" Trading System](#)
- [Smoothed RSI Buy Signals](#)
- [Ultimate plus](#)
- [Varexlist](#)

**More information:**

[Updated on–line reference](#)



**UNCISSUES****Composites****– unchanged issues**

(AFL 1.2)

**SYNTAX**     *uncissues()***RETURNS**     ARRAY**FUNCTION**     Returns the number of unchanged issues for a given market (the one that currently analysed stock belongs to)**EXAMPLE**     *uncissues()***SEE ALSO****References:**

The **UNCISSUES** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**UNCVOLUME**  
**– unchanged issues volume****Composites**  
(AFL 1.2)**SYNTAX**     *uncvolume()***RETURNS**    ARRAY**FUNCTION**    Returns the volume of unchanged issues for a given market (the one that currently analysed stock belongs to)**EXAMPLE**     *uncvolume()***SEE ALSO****References:**

The **UNCVOLUME** function is used in the following formulas in AFL on–line library:

**More information:**

[Updated on–line reference](#)

**VALUEWHEN**

Trading system toolbox

– get value of the array when condition met

(AFL 1.1)

**SYNTAX**      *valuewhen*(*EXPRESSION*, *ARRAY*, *n* = 1)**RETURNS**    ARRAY**FUNCTION**    Returns the value of the ARRAY when the EXPRESSION was true on the *n* –th most recent occurrence. Note: this function allows also 0 and negative values for *n* – this enables referencing future**EXAMPLE**      *valuewhen*( *cross*( *close*, *ma*(*close*,5) ) ,*macd*(), 1)**SEE ALSO****References:**

The **VALUEWHEN** function is used in the following formulas in AFL on–line library:

- ['D/9E H'](#)
- ['D/9E H'](#)
- [Andrews PitchforkV3.3](#)
- [AR\\_Prediction.afl](#)
- [Auto–Optimization Framework](#)
- [Automatic Trend–line](#)
- [Baseline Relative Performance Watchlist charts V2](#)
- [Bollinger band normalization](#)
- [Bullish Percent Index 2004](#)
- [Candle Stick Analysis](#)
- [Candle Stick Demo](#)
- [CCI\(20\) Divergence Indicator](#)
- [Chandelier Exit](#)
- [Cole](#)
- [Cycle Highlighter](#)
- [Cycle Highlighter \(auto best–fit\)](#)
- [Date\\_To\\_Num\(\), Time\\_To\\_Num\(\)](#)
- [Divergence indicator](#)
- [Divergences](#)
- [Double top detection](#)
- [FirstBarIndex\(\), LastBarIndex\(\)](#)
- [Fund Screener](#)
- [Gann HiLo Indicator and System](#)
- [Gann Swing Chart](#)
- [Head &Shoulders Pattern](#)
- [MACD and histogram divergence detection](#)
- [MACD commentary](#)
- [Market Profile &Market Volume Profile](#)
- [Monthly bar chart](#)
- [Pattern Recognition Exploration](#)
- [Performance Overview](#)
- [Peterson](#)
- [PF Chart – Close – April 2004](#)
- [Pivot Finder](#)

- Plot Monthly, Weekly and Daily Moving average
- Pullback System No. 1
- Rainbow Oscillator
- Relative Strength Multichart of up to 10 tickers
- RSI of Weekly Price Array
- RSI Trendlines and Wedges
- Schiff Lines
- Steve Woods' Cum. Vol. Float + Cum. Vol. Channels
- Steve Woods' Cumulative Vol. Percentage Indicator
- Stochastic Divergence, negative
- Stochastic Divergence, positive
- Stochastic Divergences, PDI, NDI
- Stochastic of Weekly Price Array
- Stochastics Trendlines
- Support and Resistance
- Support Resistance levels
- TD sequential
- The Fibonacci behavior
- Time Frame Weekly Bars
- Trend Detection
- Trend exploration with multiple timeframes
- Trigonometric Fit – TrigFit with AR for cos / sin
- Using From and To dates from Auto Analysis in Code
- Visualization of stoploses and profit in chart
- Volatility Quality Index
- Weekly chart
- Weekly Trend in Daily Graph
- Williams Alligator system
- Zig Zag Indicator with Valid Entry and Exit Points

**More information:**

Updated on–line reference

**VARGET****– gets the value of dynamic variable****Miscellaneous functions**

(AFL 2.60)

**SYNTAX**      **VarGet( "varname" )****RETURNS**      ARRAY or NUMBER

**FUNCTION**      Gets the value of dynamic variable.  
 Returns the NUMBER or ARRAY depending on type of underlying variable.

*Dynamic variables* are variables that are named dynamically, typically by creating a variable name from a static part and a variable part. For example, the following example dynamically constructs the variable name from a variable prefix and a static suffix.

**EXAMPLE**      `for( i = 1; i < 10; i++ )  
 {  
   Plot( VarGet( "C"+i ), "C"+i, colorRed );  
 }`

**SEE ALSO**      [VarSet\(\)](#) function , [VarGetText\(\)](#) function , [VarSetText\(\)](#) function

**References:**

The **VarGet** function is used in the following formulas in AFL on–line library:

- [AFL Timing functions](#)
- [Candle Identification](#)
- [Ranking Ticker WatchList](#)
- [TWS auto–export Executions–file parser](#)

**More information:**

[Updated on–line reference](#)

**VARGETTEXT****Miscellaneous functions****– gets the text value of dynamic variable**

(AFL 2.80)

**SYNTAX**      **VarGetText( "varname" )****RETURNS**      STRING

**FUNCTION**      Gets the text (string) value of dynamic variable.  
Similar to VarGet but always returns always string values (if underlying variable has different type it is converted to string) Allows for example appending to text variable no matter if it is defined earlier or not as shown in the example below

*Dynamic variables* are variables that are named dynamically, typically by creating a variable name from a static part and a variable part. For example, the following example dynamically constructs the variable name from a variable prefix and a static suffix.

**EXAMPLE**      `Title = VarGetText( "Title" ) + "something";  
// above will work correctly regardless of whenever title was  
defined earlier or not`

**SEE ALSO**      [VarGet\(\)](#) function , [VarSet\(\)](#) function

**References:**

The **VarGetText** function is used in the following formulas in AFL on–line library:

- [Calculate composites for tickers in list files](#)
- [DateNum\\_DateStr](#)
- [Ranking and sorting stocks](#)
- [TWS auto–export Executions–file parser](#)

**More information:**

[Updated on–line reference](#)

**VARSET****Miscellaneous functions****– sets the value of dynamic variable**

(AFL 2.60)

**SYNTAX**      **VarSet( "varname", value )****RETURNS**      NUMBER**FUNCTION**      Sets the value of dynamic variable. Returns 1 on success, 0 on failure.

*Dynamic variables* are variables that are named dynamically, typically by creating a variable name from a static part and a variable part. The following example dynamically constructs the variable name from a variable prefix and a static suffix.

**EXAMPLE**      `for( i = 1; i < 10; i++ )`  
                   `{`  
                   `VarSet( "C"+i, Ref( C, -i ) );`  
                   `}`

`// creates variables C1, C2, C3, C4, ..., C10 equal to Ref( C, -1`  
`), Ref( C, -2 ), ..., Ref( C, -10 )`  
`// respectively`

**SEE ALSO**      `VarGet()` function , `VarGetText()` function , `VarSetText()` function**References:**

The **VarSet** function is used in the following formulas in AFL on–line library:

- [AFL Timing functions](#)
- [Candle Identification](#)
- [Ranking and sorting stocks](#)
- [Ranking Ticker WatchList](#)
- [TWS auto–export Executions–file parser](#)

**More information:**

[Updated on–line reference](#)

**VARSETTEXT****Miscellaneous functions****– sets dynamic variable of string type**

(AFL 2.80)

**SYNTAX**      *VarSetText( "varname", "valuetext" )***RETURNS**     STRING

**FUNCTION**     Sets the text (string) value of dynamic variable.  
Similar to VarSet but allows to assign string (text) instead of number/array.

*Dynamic variables* are variables that are named dynamically, typically by creating a variable name from a static part and a variable part. For example, the following example dynamically constructs the variable name from a variable prefix and a static suffix.

**EXAMPLE**

**SEE ALSO**     [VarGetText\(\)](#) function , [VarGet\(\)](#) function , [VarSet\(\)](#) function

**References:**

The **VarSetText** function is used in the following formulas in AFL on–line library:

- [Calculate composites for tickers in list files](#)
- [DateNum\\_DateStr](#)
- [Ranking and sorting stocks](#)
- [TWS auto–export Executions–file parser](#)

**More information:**

[Updated on–line reference](#)



**VERSION****Miscellaneous functions****– get version info**

(AFL 1.9)

**SYNTAX**      *version(minrequired = 0)***RETURNS**      NUMBER

**FUNCTION**      Returns the AmiBroker version number as float ( 3.90 for example ). Additionally when you specify Version( 4.0 ) AmiBroker will issue an error message when running the formula on AB earlier than 4.0 :)

**EXAMPLE**      *version( 3.90 );*

**SEE ALSO****References:**

The **VERSION** function is used in the following formulas in AFL on–line library:

- [PFChart – High/Low prices Sept2003](#)
- [PF Chart – Close – April 2004](#)

**More information:**

[Updated on–line reference](#)

**WILDERS****Moving averages, summation****– Wilder's smoothing**

(AFL 1.4)

**SYNTAX**      *wilders( ARRAY, periods )***RETURNS**     ARRAY**FUNCTION**     Calculates Wilder's average of the ARRAY using *periods* averaging range**EXAMPLE**      *wilders( close, 10 );***SEE ALSO****References:**

The **WILDERS** function is used in the following formulas in AFL on–line library:

- [AC+ acceleration](#)
- [Adaptave Zones O/B &O/S Oscillator](#)
- [Analytic RSI formula](#)
- [AO+ Momentum indicator](#)
- [Bollinger Fibonacci Bands](#)
- [DMI Spread Index](#)
- [Fund Screener](#)
- [garythompson](#)
- [garythompson](#)
- [Raw ADX](#)
- [RSI of Weekly Price Array](#)
- [The Mean RSIt](#)
- [The Mean RSIt \(variations\)](#)
- [Twiggs Money Flow](#)
- [Volume Oscillator](#)
- [Williams Alligator system](#)

**More information:**

[Updated on–line reference](#)

## WMA – weighted moving average

Moving averages, summation  
(AFL 2.0)

**SYNTAX**      *wma( ARRAY, periods )*

**RETURNS**     ARRAY

**FUNCTION**    Calculates weighted average. 5 day weighted average gives weight of 5 to the most recent quote, 4 to the previous quote, down to 1 for the 5–bar back quote. The function accepts time–variable *periods*.

**EXAMPLE**     WMA( Close, 5 )

**SEE ALSO**     MA(), EMA(), WMA(), DEMA()

### References:

The **WMA** function is used in the following formulas in AFL on–line library:

- [Chande's Trend Score](#)
- [Chandelier Exit or Advanced Trailing Stop](#)
- [Hull Moving Average](#)
- [MultiCycle 1.0](#)
- [Relative strength comparison with moving average](#)

### More information:

[Updated on–line reference](#)

**WRITEIF**

Exploration / Indicators

**– commentary conditional text output****SYNTAX**     *writeif( EXPRESSION, "TRUE TEXT", "FALSE TEXT" )***RETURNS**     STRING**FUNCTION**     If EXPRESSION evaluates to "true", then the TRUE TEXT string is displayed within the commentary. If EXPRESSION evaluates to "false", then the FALSE TEXT string is displayed.**EXAMPLE**     `writeif( c > mov(c,200,s), "The close is above the 200–period moving average.", "The close is below the 200–period moving average." )`**SEE ALSO**     [writeval\(\) function](#)**Comments:**

<p><b>Tomasz Janeczko</b>  tj --at-- amibroker.com  2004–06–12 05:56:01</p>	<p>Writelf in fact does not "write" anything. The name is misleading but it is left for easy translation of MS formulas to AmiBroker. Writelf is just "TextIf" it RETURNS string value depending on condition.</p> <p>In commentary window, statements evaluating to STRINGS on global level are displayed in the output window. However if you do the same inside the FUNCTION it is no longer in global level (it is on LOCAL, FUNCTION level).</p> <p>To display actual string in this case use PRINTF function:  <a href="http://www.amibroker.com/f?printf">http://www.amibroker.com/f?printf</a></p> <pre>function comment(indicator) { printf( "\nComment...\n" );  printf( Writelf(1, "TrueText", "FalseText") ); printf( WriteVal(indicator) + "\n" ); }</pre>
<p><b>Tomasz Janeczko</b>  2005–08–10 06:37:55</p>	<p>Please note that Writelf returns just single string representing current SelectedValue of the EXPRESSION</p>

**References:**

The **WRITEIF** function is used in the following formulas in AFL on–line library:

- ['D/9E H'](#)
- ['D/9E H'](#)
- [3 Price Break](#)
- [Adaptave Zones O/B &O/S Oscillator](#)
- [AR\\_Prediction.afl](#)

- BMTRIX Intermediate Term Market Trend Indicator
- CandleStick Comentary--Help needed
- Candlestick Commentary
- Candlestick Commentary Modified
- Candlestick Commentary--modified
- CCI(20) Divergence Indicator
- Color Display.afl
- Commodity Channel Index
- Cycle Highlighter
- Cycle Highlighter (auto best-fit)
- Double top detection
- ekeko price chart
- Elder Impulse Indicator V2
- MA Difference 20 Period
- MACD commentary
- Main price chart with Rainbow & SAR
- Performance Check
- PF Chart – Close – April 2004
- Polarized Fractal Efficiency
- prakash
- R-Squared
- Relative Strength Index
- Relative Strength Multichart of up to 10 tickers
- ROC of MACD Weekly
- RSI of Weekly Price Array
- STD\_STK Multi
- Stochastics Trendlines
- StochD\_StochK Single.afl
- Sunl>Time Left in Bar
- Trend Analysis\_Comentary
- Trend exploration with multiple timeframes
- Triangular Moving Average
- Triangular Moving Average new
- Vertical Horizontal Filter (VHF)
- Weekly Trend in Daily Graph
- Williams Alligator system
- Zig Zag Indicator with Valid Entry and Exit Points

**More information:**

Updated on-line reference

**WRITEVAL****Exploration / Indicators****– write number or value of the array**

**SYNTAX**      *WriteVal( NUMBER, format = 1.3, separator=True)*  
*WriteVal( ARRAY, format = 1.3, separator=True )*

**RETURNS**    STRING

**FUNCTION**    It is used to display the numeric value of NUMBER or ARRAY. The second parameter – *format* – allows you to control output formatting (decimal places and leading spaces). The integer part of the number controls minimum number of characters used to display the number (if you specify high number the output will be space-padded). The fractional part defines how many decimal places to display, for example 1.0 – will give you a number without fractional part at all, and 1.2 – will display two digits past the decimal point

There is also a special format constant `formatDateTime` that allows to print date/time returned by `DateTime()` function formatted according to Windows regional settings. Third parameter *separator* (true by default) controls if thousand separator is added or not. Thousands separator is definable in Tools->Preferences->Misc.

Note: **NumToStr** is a synonym for **WriteVal** function and **NumToStr** is preferred in new coding.

**EXAMPLE**    1. Simple use (no custom format)

```
WriteVal( StochK(39) - StochK(12) );
```

2. Display rate of change with 2 decimal digits and % appened to the end

```
WriteVal( ROC( Close, 20 ), 1.2 ) + "%";
```

3. Display date/time according to regional settings

```
WriteVal( DateTime(), formatDateTime );
```

**SEE ALSO**    `WRITEIF()` function , `DATETIME()` function , `NumToStr()` function

**Comments:**

<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2004-06-12 05:53:06	WriteVal always returns *one* value of the array (not arrays of values). In almost all cases this is LastValue of the array but in indicators it is "selected value" – the one that is selected by the vertical line.
<b>Tomasz Janeczko</b> tj --at-- amibroker.com 2004-06-12 05:54:45	The name WriteVal() is here because people coming from Metastock wanted easy translation from MS that has WriteVal function too. Better name for it is Num2Str.

**References:**

The **WriteVal** function is used in the following formulas in AFL on–line library:

- 'D/9E H'
- 'D/9E H'
- AccuTrack
- ADX Indicator – Colored
- AFL Example
- AFL Example – Enhanced
- Alert Output As Quick Rewiev
- Alpha and Beta and R\_Squared Indicator
- Aroon Indicators
- AR\_Prediction.afl
- Baseline Relative Performance Watchlist charts V2
- BB squeeze
- Bollinger – Keltner Bands
- Bull Fear / Bear Fear
- Bullish Percent Index 2 files combined
- Bullish Percent Index 2004
- CandleStick Comentary—Help needed
- CCI(20) Divergence Indicator
- Chande Momentum Oscillator
- Cole
- Color Coded Short Term Reversal Signals
- colored CCI
- COMBO
- Commodity Channel Index
- Count Tickers in Watchlist
- Cycle Highlighter
- Cycle Highlighter (auto best–fit)
- De Mark's Range Projection
- Dinapoli Guru Commentary
- Elder safe Zone Long + short
- Elder's SafeZone Stop
- ElderSafeZoneStopLong
- ElderSafeZoneStopShort
- Fibonacci Moving averages
- Futures – Dollar Move Indicator
- Futures – Dollar Move Today Indicator
- Gordon Rose
- hassan
- IntraDay Open Marker
- Larry William's Volatility Channels
- MA Difference 20 Period
- MACD and histogram divergence detection
- MACD commentary
- MACD Histogram – Change in Direction
- MACD indicator display
- Main price chart with Rainbow & SAR
- Modified Darvas Box
- Monthly bar chart

- Monthly Coppock Guide
- Multiple sinus noised
- N-period candlesticks (time compression)
- Option Calls, Puts and days till third friday.
- PFchart with range box sizes
- Parabolic SAR in JScript
- Parabolic SAR in VBScript
- Performance Check
- Performance Overview
- Peterson
- PF Chart – Close – April 2004
- Pivot Finder
- Pivot Point and Support and Resistance Points
- Plot Monthly, Weekly and Daily Moving average
- Position Sizing and Risk Price Graph
- Position Sizing and Risk Price Graph – 2
- prakash
- Probability Calculator
- QP2 Float Analysis
- Rainbow Charts
- Rainbow Oscillator
- Rea Time Daily Price Levels
- Relative Strength Index
- Relative Strength Multichart of up to 10 tickers
- ROC of MACD Weekly
- RSI of Weekly Price Array
- RSIS
- Shares To Buy Price Graph
- STD\_STK Multi
- Steve Woods' Float Channel Lines
- Stochastics Trendlines
- StochD\_StochK Single.afl
- Support Resistance levels
- The D\_oscillator
- tomy\_frenchy
- Trend Analysis\_Comentary
- Triangular Moving Average
- Triangular Moving Average new
- Trigonometric Fit – TrigFit with AR for cos / sin
- TRIX
- Tushar Chande's Projected Range
- Using From and To dates from Auto Analysis in Code
- Volume Oscillator
- Weinberg's The Range Indicator
- William's % R
- Williams %R with 9 period signal line
- Williams Alligator system

#### More information:

Updated on-line reference





**YEAR**  
– year**Date/Time**  
(AFL 1.4)**SYNTAX**     *year()***RETURNS**     ARRAY**FUNCTION**     Returns the array with years (full four digits 1900–....)**EXAMPLE**     writeval( year() );**SEE ALSO****References:**

The **YEAR** function is used in the following formulas in AFL on–line library:

- [AB system full automation scripts](#)
- [Days to Third Friday](#)
- [End Of Year Trading](#)
- [Expiry Thursday for Indian markets](#)
- [Export Intraday Data](#)
- [Luna Phase](#)
- [LunarPhase](#)
- [Monthly bar chart](#)
- [N–period candlesticks \(time compression\)](#)
- [Relative Strength Multichart of up to 10 tickers](#)

**More information:**

[Updated on–line reference](#)

**ZIG****Basic price pattern detection****– zig–zag indicator**

(AFL 1.1)

**SYNTAX**      *zig*(ARRAY, *change* )**RETURNS**     ARRAY

**FUNCTION**     Calculates the minimum % *change* Zig Zag indicator. **Caveat:** this function is based on Zig–Zag indicator and may look into the future – this means that you can get unrealistic results when back testing trading system using this indicator. This function is provided rather for pattern and trend recognition formulas.

**EXAMPLE**     zig(close,5)**SEE ALSO****References:**

The **ZIG** function is used in the following formulas in AFL on–line library:

- [Andrews PitchforkV3.3](#)
- [Bollinger band normalization](#)
- [Divergence indicator](#)
- [Indicator Explorer \(ZigZag\)](#)
- [QP2 Float Analysis](#)
- [Schiff Lines](#)
- [The Fibonacci behavior](#)
- [Volatility Quality Index](#)
- [Zig Explorer](#)
- [Zig Zag Indicator with Valid Entry and Exit Points](#)

**More information:**

[Updated on–line reference](#)

**\_DEFAULT\_NAME****Exploration / Indicators**  
(AFL 2.70)**– retrieve default name of the plot****SYNTAX**     **\_DEFAULT\_NAME()****RETURNS**     STRING**FUNCTION**     This function returns the default name of plot in the drag–drop section. The default name consists of section name and comma separated list of values of numeric parameters defined in given section.**EXAMPLE**

```
_SECTION_BEGIN( "MA1" );
P = ParamField( "Price field" );
Periods = Param( "Periods", 15, 2, 200, 1, 10 );
Plot( MA( P, Periods ), _DEFAULT_NAME(), ParamColor( "Color",
colorCycle ), ParamStyle( "Style" ) );
_SECTION_END();
```

      \_DEFAULT\_NAME will evaluate to "MA1(Close,15)" string.

**SEE ALSO**     [\\_SECTION\\_BEGIN\(\)](#) function , [\\_SECTION\\_NAME\(\)](#) function , [\\_SECTION\\_END\(\)](#) function**References:**

The **\_DEFAULT\_NAME** function is used in the following formulas in AFL on–line library:

- [AFL Example](#)
- [AFL Example – Enhanced](#)
- [Dave Landry PullBack Scan](#)
- [DPO with shading](#)
- [Elder Triple Screen Trading System.](#)
- [Elder's Market Thermometer](#)
- [Fibonacci Moving averages](#)
- [INTRADAY HEIKIN ASHI new](#)
- [Mndahoo ADX](#)
- [Noor\\_Doodie](#)
- [ParabXO](#)
- [prakash](#)
- [Simple Momentum](#)
- [Triangular Moving Average new](#)
- [Twiggs Money Flow](#)
- [Ultimate plus](#)
- [UltraEdit editor highlight wordfile](#)
- [William's % R](#)
- [Williams %R Exploration](#)
- [ZigZag filter rewritten from scratch in AFL](#)
- [\\_Volume](#)

**More information:**[Updated on–line reference](#)

**\_N****Exploration / Indicators**  
(AFL 2.1)**– no text output****SYNTAX**     **\_N( string )****RETURNS**     nothing**FUNCTION**     Protects from printing string to the commentary output window**EXAMPLE**     `_N( ticker = name() );` // thanks to \_N function ticker symbol is not printed**SEE ALSO**     [ENABLETEXTOUTPUT\(\)](#) function**References:**The **\_N** function is used in the following formulas in AFL on–line library:

- ['D/9E H'](#)
- ['D/9E H'](#)
- ['DE\\*H37href='http://www.amibroker.com/library/detail.php?id=325target='\\_blank'>'R' Channel](#)
- [10–20 Indicator](#)
- [30 Week Hi Indicator – Display](#)
- [52 Week New High–New Low Index](#)
- [AB system full automation scripts](#)
- [abosliman2005m](#)
- [AccuTrack](#)
- [Adaptive Laguerre Filter, from John Ehlers](#)
- [ADXbuy](#)
- [AFL Example](#)
- [AFL Example – Enhanced](#)
- [Alert Output As Quick Rewiev](#)
- [Alpha and Beta and R\\_Squared Indicator](#)
- [Andrews Pitchfork](#)
- [Andrews PitchforkV3.3](#)
- [Aroon Indicators](#)
- [AR\\_Prediction.afl](#)
- [ATR Study](#)
- [Auto–Optimization Framework](#)
- [Automatic Trend–line](#)
- [Baseline Relative Performance Watchlist charts V2](#)
- [Black Scholes Option Pricing](#)
- [BMTRIX Intermediate Term Market Trend Indicator](#)
- [Bollinger band normalization](#)
- [Bollinger Band Width](#)
- [Bottom Trader](#)
- [Bull Fear / Bear Fear](#)
- [Bullish Percent Index 2 files combined](#)
- [Bullish Percent Index 2004](#)
- [Calculate composites for tickers in list files](#)
- [CAMSLIM Cup and Handle Pattern AFL](#)
- [Candle Identification](#)
- [Candle Pattern Function](#)
- [Candle Stick Analysis](#)

- Candle Stick Demo
- CandleStick Comentary---Help needed
- Candlestick Commentary
- Candlestick Commentary Modified
- Candlestick Commentary--modified
- CCI(20) Divergence Indicator
- Chandelier Exit
- Cole
- Color Display.afl
- Count Tickers in Watchlist
- crMathLib
- Cycle Highlighter
- Cycle Highlighter (auto best-fit)
- danningham penetration
- DateNum\_DateStr
- Date\_To\_Num(), Time\_To\_Num()
- Dave Landry PullBack Scan
- Divergence indicator
- Divergences
- DMI Spread Index
- Double top detection
- Ed Seykota's TSP: EMA Crossover System
- Ed Seykota's TSP: Support and Resistance
- ekeko price chart
- EKEKO SAR-MF
- Elder Impulse Indicator V2
- Elder Ray – Bull Bear
- Elder safe Zone Long + short
- Elder Triple Screen Trading System.
- Elder's Market Thermometer
- EMA Crossover
- End Of Year Trading
- Evaluating Candle Patterns in a trading system
- Export Intraday Data
- FastStochK FullStochK-D
- Fibonacci Moving averages
- FirstBarIndex(), LastBarIndex()
- Follow the Leader
- Force index
- Fund Screener
- Future MA Projection
- Gann HiLo Indicator and System
- Gann Swing Chart
- Gordon Rose
- half-automated Trading System
- Head &Shoulders Pattern
- Hilbert Study
- Historical Volatility Scan – 6/100
- Historical Volatility Scan – 50 Day
- Index of 30 Wk Highs Vs Lows
- Indicator Explorer (ZigZag)
- INTRADAY HEIKIN ASHI new

- Luna Phase
- LunarPhase
- MACD and histogram divergence detection
- MACD commentary
- MACD Histogram – Change in Direction
- MACD indicator display
- Market Profile & Market Volume Profile
- Modified Darvas Box
- Monthly bar chart
- Moving Average "Crash" Test
- MultiCycle 1.0
- Multiple sinus noised
- Negative ROC Exploration
- nifty
- Noor\_Doodie
- nth ( 1 – 8 ) Order Polynomial Fit
- Option Calls, Puts and days till third friday.
- PFChart – High/Low prices Sept2003
- PFchart with range box sizes
- Parabolic SAR in VBScript
- Pattern – Rectangle Base Breakout on High Vol
- pattern correlation
- Pattern Recognition Exploration
- Performance Check
- Performance Overview
- Peterson
- PF Chart – Close – April 2004
- Pivot Finder
- Pivots for Intraday Forex Charts
- Plot Monthly, Weekly and Daily Moving average
- Positive ROC Exploration
- prakash
- Price Persistency
- Projection Oscillator
- Pullback System No. 1
- R-Squared
- Rainbow Oscillator
- Ranking and sorting stocks
- Ranking Ticker WatchList
- Relative Strength
- Relative Strength Index
- Relative Strength Multichart of up to 10 tickers
- Renko Chart
- RSI "based" Trading System
- RSI Double-Bottom
- RSI of volume weighted moving average
- RSI of Weekly Price Array
- RSI Trendlines and Wedges
- RUTVOL timing signal with BB Scoring routine
- Sainath Sidgiddi
- SAR-ForNextBarStop
- Schiff Lines

- Sector Tracking
- SectorRSI
- Simple Candle Exploration
- Smoothed RSI Buy Signals
- STD\_STK Multi
- Steve Woods' Cum. Vol. Float + Cum. Vol. Channels
- Steve Woods' Cumulative Vol. Percentage Indicator
- Stochastic Divergence, negative
- Stochastic Divergence, positive
- Stochastic Divergences, PDI, NDI
- Stochastic of Weekly Price Array
- Stochastics Trendlines
- StochD\_StochK Single.afl
- Stops Implementation in AFS
- Strength and Weakness
- SUPER PIVOT POINTS
- Support and Resistance
- Support Resistance levels
- TAZ Trading Method Exploration
- TD sequential
- The D\_oscillator
- The Fibonacci behavior
- Three Line Break – TLB
- Time Frame Weekly Bars
- Time Left in Bar
- tomy\_frenchy
- Trend Analysis\_Comentary
- Trend Detection
- Trend exploration with multiple timeframes
- Trending Ribbon
- Triangle exploration using PFChart
- Triangular Moving Average new
- Trigonometric Fit – TrigFit with AR for cos / sin
- Twiggs Money Flow
- TWS auto-export Executions-file parser
- Ultimate plus
- UltraEdit editor highlight wordfile
- Using From and To dates from Auto Analysis in Code
- Visualization of stoploses and profit in chart
- Volatility Quality Index
- Volatility System
- Volume – compared with Moving Avg (100%)
- Weekly chart
- Weekly Trend in Daily Graph
- Weighted Index
- Williams %R Exploration
- Williams Alligator system
- WLBuildProcess
- Zig Zag
- Zig Zag Indicator with Valid Entry and Exit Points
- ZigZag filter rewrited from scratch in AFL
- \_Volume



**More information:**

[Updated on-line reference](#)

**\_PARAM\_VALUES****Exploration / Indicators****– retrieve param values string**

(AFL 2.70)

**SYNTAX**     **\_PARAM\_VALUES()****RETURNS**     STRING

**FUNCTION**     \_PARAM\_VALUES retrieves the values of the parameters defined in current drag-drop section. It works the same as \_DEFAULT\_NAME except that no section name is included (so only the list of parameter values is returned).

**EXAMPLE**

**SEE ALSO**     [\\_DEFAULT\\_NAME\(\)](#) function

**References:**

The **\_PARAM\_VALUES** function is used in the following formulas in AFL on-line library:

- [Elder Triple Screen Trading System](#).
- [INTRADAY HEIKIN ASHI new](#)
- [prakash](#)

**More information:**

[Updated on-line reference](#)

**\_SECTION\_BEGIN****Exploration / Indicators****– section begin marker**

(AFL 2.70)

**SYNTAX**     **\_SECTION\_BEGIN( "section name" )****RETURNS**     NOTHING**FUNCTION**     Marks beginning of the drag-drop section.

IMPORTANT: "section name" must be a constant, literal string, enclosed in double quotation marks. **You must NOT use variable here.**

**EXAMPLE****SEE ALSO****References:**

The **\_SECTION\_BEGIN** function is used in the following formulas in AFL on-line library:

- 'D/9E H'
- 'D/9E H'
- 10–20 Indicator
- AFL Example
- Bollinger Band Width
- Candle Identification
- Candle Stick Demo
- Chandelier Exit
- DateNum\_DateStr
- Dave Landry PullBack Scan
- Elder Impulse Indicator V2
- Elder Ray – Bull Bear
- Elder safe Zone Long + short
- Elder Triple Screen Trading System.
- Elder's Market Thermometer
- FastStochK FullStochK–D
- Fibonacci Moving averages
- Force index
- Future MA Projection
- Historical Volatility Scan – 6/100
- Historical Volatility Scan – 50 Day
- INTRADAY HEIKIN ASHI new
- Luna Phase
- MACD and histogram divergence detection
- MACD Histogram – Change in Direction
- Modified Darvas Box
- Noor\_Doodie
- Option Calls, Puts and days till third friday.
- prakash
- RSI "based" Trading System
- RSI of volume weighted moving average
- Smoothed RSI Buy Signals
- SUPER PIVOT POINTS

- [Trending Ribbon](#)
- [Triangular Moving Average new](#)
- [Twiggs Money Flow](#)
- [Ultimate plus](#)
- [UltraEdit editor highlight wordfile](#)
- [Williams %R Exploration](#)
- [ZigZag filter rewritten from scratch in AFL](#)
- [\\_Volume](#)

**More information:**

[Updated on-line reference](#)

**\_SECTION\_END****Exploration / Indicators**  
(AFL 2.70)**– section end marker****SYNTAX**     **\_SECTION\_END()****RETURNS**     NOTHING**FUNCTION**     marks end of drag–drop section**EXAMPLE****SEE ALSO****References:**

The **\_SECTION\_END** function is used in the following formulas in AFL on–line library:

- 'D/9E H'
- 'D/9E H'
- 10–20 Indicator
- AFL Example
- Bollinger Band Width
- Candle Identification
- Candle Stick Demo
- Chandelier Exit
- Dave Landry PullBack Scan
- Elder Impulse Indicator V2
- Elder Ray – Bull Bear
- Elder safe Zone Long + short
- Elder Triple Screen Trading System.
- Elder's Market Thermometer
- FastStochK FullStochK–D
- Fibonacci Moving averages
- Force index
- Future MA Projection
- Historical Volatility Scan – 6/100
- Historical Volatility Scan – 50 Day
- INTRADAY HEIKIN ASHI new
- Luna Phase
- MACD and histogram divergence detection
- MACD Histogram – Change in Direction
- Modified Darvas Box
- Noor\_Doodie
- Option Calls, Puts and days till third friday.
- prakash
- RSI "based" Trading System
- RSI of volume weighted moving average
- Smoothed RSI Buy Signals
- SUPER PIVOT POINTS
- Trending Ribbon
- Triangular Moving Average new
- Twiggs Money Flow
- Ultimate plus

- [UltraEdit editor highlight wordfile](#)
- [Williams %R Exploration](#)
- [ZigZag filter rewritten from scratch in AFL](#)
- [\\_Volume](#)

**More information:**

[Updated on-line reference](#)

**\_SECTION\_NAME****Exploration / Indicators**  
(AFL 2.70)**– retrieve current section name****SYNTAX**     **\_SECTION\_NAME()****RETURNS**     STRING**FUNCTION**     The function that gives the name of the drag-drop section (given in previous \_SECTION\_BEGIN call).**EXAMPLE****SEE ALSO**     [\\_SECTION\\_BEGIN\(\)](#) function**References:**

The **\_SECTION\_NAME** function is used in the following formulas in AFL on-line library:

- [Elder Triple Screen Trading System](#).
- [UltraEdit editor highlight wordfile](#)

**More information:**[Updated on-line reference](#)

## **\_TRACE** – print text to system debug viewer

**Miscellaneous functions**  
(AFL 2.4)

**SYNTAX**     **\_TRACE("string")**

**RETURNS**     NOTHING

**FUNCTION**     Write debug messages from AFL code to system debug viewer. (it calls internally OutputDebugString Win API function). To view debug messages you have to run DebugView freeware program from <http://www.sysinternals.com/Utilities/DebugView.html>

**EXAMPLE**     `_TRACE("This is a test");`  
                   `_TRACE("This is selected value of close: " + Close );`  
                   `_TRACE("This is first element of close array: " + Close[ 0 ] );`

### **SEE ALSO**

#### **References:**

The **\_TRACE** function is used in the following formulas in AFL on–line library:

- [Calculate composites for tickers in list files](#)
- [CCI\(20\) Divergence Indicator](#)
- [Ed Seykota's TSP: EMA Crossover System](#)
- [Ed Seykota's TSP: Support and Resistance](#)
- [lastNDaysBeforeDate](#)
- [Pivot Finder](#)
- [WLBuildProcess](#)

#### **More information:**

[Updated on–line reference](#)

## **AFL Error List**

- Error 1. Operation not allowed. Operator/operand type mismatch.
- Error 2. Incorrect type of argument. Expecting number or array here.
- Error 3. Type mismatch, Unary minus operator requires number or array
- Error 4. Incorrect type of argument(s). Expecting number here.
- Error 5. Argument #1 has incorrect type (the function expects different argument type here)
- Error 6. Condition in IF, WHILE, FOR statements has to be Numeric or Boolean type. You can not use array here
- Error 7. Condition in IF, WHILE, FOR statements has to be Numeric or Boolean type. You can not use STRING here



- Error 8. Type mismatch, the value assigned to the array element has to be a number. You can not use array...
- Error 9. Array subscript has to be a number
- Error 10. Subscript out of range. You must not access array elements outside 0..(BarCount-1) range
- Error 11. Subscript operator [] requires array or number type. String can not be used here.
- Error 12. Subscript operator [] requires array or number type.
- Error 13. Endless loop detected in WHILE loop
- Error 14. Endless loop detected in DO-WHILE loop
- Error 15. Endless loop detected in FOR loop
- Error 16. Too many arguments
- Error 17. Missing arguments
- Error 18. COM object variable is not initialized or has invalid type (valid COM object handle required)
- Error 19. COM method/function 'function name' call failed.
- Error 20. COM Method/function '%s' does not exist
- Error 21. Relative strength base symbol not found
- Error 22. Bad 'format' argument type – format has to be a number (not array)
- Error 23. GetExtraData call failed
- Error 24. This formula requires AmiBroker version '...' (or higher).
- Error 25. SetVariable() called from plug in was not successful. Identifier is already used in a different context.
- Error 26. File handle passed to the function is invalid (equal to zero).
- Error 27. Invalid number of arguments passed to Call Function (..) from plugin DLL
- Error 28. Out of memory
- Error 29. Variable used without having been initialized.
- Error 30. Syntax error
- Error 31. Syntax error, expecting 'list of tokens'

- Error 32. Syntax error, probably missing semicolon at the end of the previous line
- Error 33. The identifier is already in use. You attempted to assign value to the identifier representing a function.
- Error 34. The identifier is already in use. You attempted to define the function that has the same identifier as global variable.
- Error 35. Shift+BREAK pressed. Loop terminated.
- Error 36. N-th argument of the function call has no value set
- Error 37. Unsupported field in SetOptions
- Error 38. Unsupported field in GetOptions
- Error 39. CategoryAddSymbol: Setting sector is unsupported, set industry instead
- Error 40. CategoryRemoveSymbol: Removing from sector is unsupported, remove from industry instead
- Error 41. Unsupported field in GetRTData
- Error 42. #include failed because the file does not exist: 'filename' (current working directory is '...')
- Error 43. Variable stops are not supported in Rotational Trading mode
- Error 44. SectorID() is outside 0..63 range.
- Error 45. Failed to launch trading interface
- Error 46. Missing comma
- Error 47. Exception occurred during AFL formula execution
- Error 48. N-volume bar compressed data longer than base time frame. Use higher compression factor.
- Error 49. Optimization parameter name must not be empty.
- Error 50. Optimization parameter minimum value must be less than or equal to maximum and step parameter needs to be greater than zero.

#### **Error 1. Operation not allowed. Operator/operand type mismatch.**

An arithmetic, string, logical or comparison operator is being used with an invalid data type. This error would occur, for example, if you were to attempt to multiply two string values.

```
a = "x" * 5; // wrong, can not multiply string by number
b = "x" - "y"; // wrong, can not subtract strings
```

```
z = "x" + "y"; // correct, concatenation of strings is OK
```

### **Error 2. Incorrect type of argument. Expecting number or array here.**

This occurs when calling single-argument mathematical function like sin() which accepts only numbers and arrays,  
but the user specified string for example.

```
x= sin("test"); // sin requires number or array
```

### **Error 3. Type mismatch, Unary minus operator requires number or array**

Occurs when trying to apply unary minus operator to strings.

```
text2 = - "test"; // can not use unary minus (negation) to texts
```

### **Error 4. Incorrect type of argument(s). Expecting number here.**

This occurs in ApplyStop function when Type, Mode, ExitAtStop or Volatile parameter is an array

```
// wrong - stop mode (percent/point) can not be array
ApplyStop( stopTypeLoss, IIf( C > 0, stopModePercent, stopModePoint ), 5 );
```

### **Error 5. Argument #1 has incorrect type (the function expects different argument type here)**

This error occurs when argument passed during function call has invalid type. For example when you pass string instead of array

```
MA( "test", 5 ); // wrong, Moving average expects array as first argument
AddColumn( "Test", "Caption" ); // wrong, AddColumn expects array as first argument

AddTextColumn( "Test", "Caption" ); // correct, AddTextColumn expects text
```

### **Error 6. Condition in IF, WHILE, FOR statements has to be Numeric or Boolean type. You can not use array here, please use [] (array subscript operator) to access array elements**

The if keyword executes statement1 if expression is true (nonzero); if else is present and expression is false (zero), it executes statement2. After executing statement1 or statement2, control passes to the next statement. Expression must be boolean ( True/False) type (so it CANNOT be ARRAY because there would be no way to decide whether to execute statement1 or not, if for example array was:  
[True,True,False,.....,False,True] )

```
if( expression )
    statement1;
else
    statement2;
```

### **EXAMPLE**

```

if( Close > Open ) // WRONG
    Color = colorGreen; //statement 1
else
    Color = colorRed; //statement 2

Plot(Close, "Colored Price", Color, styleCandle);

```

The above example is wrong, as both Open and Close are arrays and such expression as Close > Open is also an ARRAY. The solution depends on the statement. It is either possible to implement it on bar-by-bar basis, with use of FOR loop:

```

for( i = 0; i < BarCount; i++ )
{
    if( Close[ i ] > Open[ i ] ) // CORRECT
        Color[ i ] = colorGreen;
    else
        Color[ i ] = colorRed;
}

Plot( Close, "Colored Price", Color, styleCandle );

```

It is also possible in this case to use IIf() function:

```

Color = IIf( Close > Open, colorGreen, colorRed ); // ALSO CORRECT - working
directly on arrays
Plot( Close, "Colored Price", Color, styleCandle );

```

**Error 7. Condition in IF, WHILE, FOR statements has to be Numeric or Boolean type. You can not use STRING here.**

Occurs when you attempt to use string as a condition in if/while/for statements.

For example:

```

if( "text" ) // incorrect
{
    // do something
    x = 1;
}

```

The condition in if/while/for should evaluate to true/false:

```

if( "text" != "someothertext" ) // correct, != (not equal to) operator gives
true/false value
{
    // do something
    x = 1;
}

```

**Error 8. Type mismatch, the value assigned to the array element has to be a number. You can not use array on the right-side of this assignment.**

Occurs on attempt to assign entire array to single element of another array

```
test = 0;
for( i = 0; i < BarCount; i++ )
{
    test[ i ] = Close ; // wrong, single array element can take only one value,
                        // not array

    test[ i ] = Close[ i ]; // correct
}
```

**Error 9. Array subscript has to be a number**

You can use only numbers as array subscripts, strings and arrays are not accepted:

```
table[ 1 ] = 10; // correct
table[ "text" ] = 10; // incorrect
table[ Close ] = 10; // incorrect
```

**Error 10. Subscript out of range. You must not access array elements outside 0..(BarCount-1) range**

Occurs when you attempt to access array elements with subscripts below 0 (zero) or above BarCount-1.

```
// incorrect
for( bar = 0; bar < BarCount; bar++ )
{
    a[ bar ] = C[ bar - 1 ]; // when i == 0 we are accessing C[-1] which is wrong
}

// correct
for( bar = 0; bar < BarCount; bar++ )
{
    if( bar > 0 )
        a[ bar ] = C[ bar - 1 ]; // only access C[ i - 1 ] when i is greater
    // than zero
    else
        a[ bar ] = C[ 0 ];
}
```

One of most common mistakes is using hard coded upper limit of for loop and assuming that all symbols have enough quotes.

For example:

```
MyPeriod = 10;
```

```
for( i = 0; i < MyPeriod; i++ ) // WRONG ! this assumes that you always have at
least 10 quotes !
{
    // ... do something
}
```

This will always fail on symbols that have less quotes than 10 and it may also fail if you zoom your chart in so less than 10 quotes are actually displayed on the chart.

To ensure error-free operation you must always check for upper index being LESS than BarCount, like shown in the code below:

```
MyPeriod = 10;
```

```
for( i = 0; i < MyPeriod AND i < BarCount; i++ ) // CORRECT - added check for
upper bound
{
    // ... do something
}
```

Alternatively you can enter the loop only when there are enough bars, like shown in this code:

```
MyPeriod = 10;
```

```
if( MyPeriod <= BarCount ) // CORRECT - check if there are enough bars to run the
loop
{
    for( i = 0; i < MyPeriod; i++ )
    {
        // ... do something
    }
}
```

#### **Error 11. Subscript operator [] requires array or number type. String can not be used here.**

Occurs when you attempt to use subscript operator [] on strings, for example:

```
tt = "Test";

x = tt[ 0 ];
```

#### **Error 12. Subscript operator [] requires array or number type.**

Occurs when subscript operator [] is applied to some other unsupported type (such as COM object dispatch):

```
a=CreateObject( "Broker.Application" );
```

```
b = a[0];
```

### Error 13. Endless loop detected in WHILE loop

Occurs when AFL engine detect the *while* loop that never ends (the detection is based on number of iterations, AmiBroker simply counts the number of iterations and if it exceeds the threshold limit set in **Tools→Preferences→AFL: Endless loop detection threshold** – by default 100000 iterations) it displays this message). Example:

```
i = 0;

while( i < 5 ) x = i; // i variable is not incremented, so the loop never ends.
```

### Error 14. Endless loop detected in DO-WHILE loop

Occurs when AFL engine detect the *do-while* loop that never ends (the detection is based on number of iterations, AmiBroker simply counts the number of iterations and if it exceeds the threshold limit set in **Tools→Preferences→AFL: Endless loop detection threshold** – by default 100000 iterations) it displays this message). Example:

```
i = 0;

do
{
    x = i;
}
while( i < 5 ); // i variable is not incremented, so the loop never ends.
```

### Error 15. Endless loop detected in FOR loop

Occurs when AFL engine detect the *for* loop that never ends (the detection is based on number of iterations, AmiBroker simply counts the number of iterations and if it exceeds the threshold limit set in **Tools→Preferences→AFL: Endless loop detection threshold** – by default 100000 iterations) it displays this message). Example:

```
for( i = 0; i < BarCount; i ) // forgotten ++ (increment operator) so the loop
never ends.
{
    x = i;
}
```

### Error 16. Too many arguments

Occurs when too many arguments were specified when calling the function. For example:

```
m = MACD( 12, 26, 3 ); // error: MACD needs only 2 parameters, but 3 are
specified
```

### Error 17. Missing arguments

Occurs when too few arguments were passed during function call. Example:

```
Plot( C, "Price" ); // too few arguments, 3rd argument is required - color of the
plot
```

### Error 18. COM object variable is not initialized or has invalid type (valid COM object handle required)

Occurs on attempt to use uninitialized variable or variable of incorrect type to call COM object methods:

```
Obj = 1; // initialize as number
Obj.Test(); // attempt to call method fails because Obj is not COM object handle
```

### Error 19. COM method/function <function name> call failed. <More info>

An OLE exception occurred during OLE/COM method/function/property call. <More info> may contain OLE exception description.

Commonly this is displayed when arguments passed to the OLE/COM method are incorrect or missing.

```
AB=CreateObject( "Broker.Application" );
AB.Import(); // <-- fails with error 19. In this case because of missing
arguments
```

### Error 20. COM Method/function '%s' does not exist

Occurs on attempt to call not existing OLE method/property. Example:

```
AB=CreateObject( "Broker.Application" );
AB.Test(); // there is no such method
```

### Error 21. Relative strength base symbol not found

RelStrength() function called with non-existing symbol as a parameter:

```
x = RelStrength( "NonExistingTicker" ); // fails because of wrong symbol
```



**Error 22. Bad 'format' argument type – format has to be a number (not array)**

Format parameter specified in AddColumn() function call should be a number (not array). Example

```
AddColumn( C, "Close", IIf( C > 10, 1.2, 1.3 ) ); // wrong, format parameter
(3rd) has to be a number
```

```
AddColumn( C, "Close", 1.3 ); // correct
```

**Error 23. GetExtraData call failed**

GetExtraData() failed (or returned no value) either because current plugin does not support GetExtraData function or field specified is not supported by the plugin:

```
x=GetExtraData("nonexistingfieldname"); // wrong field name
```

**Error 24. This formula requires AmiBroker version <...> (or higher).**

The formula is intended to be used on higher version of AmiBroker and you should upgrade in order to use it.

```
Version(9.7); // there is no version 9.70 yet :-)
```

**Error 25. SetVariable() called from plug in was not successful. Identifier is already used in a different context.**

Happens if external plugin attempts to call SetVariable with identifier that is already used for some other purpose  
(like built-in or user-defined function)

**Error 26. File handle passed to the function is invalid (equal to zero). You have to check if file handle returned by fopen is not equal to zero. If it is zero it means that file could not be opened.**

Occurs on attempt to call file write/read/close function on null file handle.

Example (incorrect):

```
fh = fopen("nonexistingfile.txt", "r"); // this file does not exist

fputs("Test", fh ); // error here, fh could be null
fclose( fh ); // wrong, fh could be null
```

Correct usage would look like this:

```
fh = fopen("nonexistingfile.txt", "r"); // this file does not exist

if( fh ) // correct, call subsequent file read/write/close only when file handle
is OK
{
    fputs("Test", fh );
    fclose( fh );
}
```

**Error 27. Invalid number of arguments passed to Call Function (..) from plugin DLL**

Occurs only when external plugin calls internal AmiBroker functions with incorrect number of arguments.

**Error 28. Out of memory**

Out-of-memory error occurred during parsing the formula (should not happen under normal circumstances)

**Error 29. Variable <name> used without having been initialized.**

You can not use (read) the variable that was not initialized first. You should assign the value to the variable before reading it.

Example (incorrect usage):

```
x = 1;
z = x + y; // wrong, y is not initialized
```

Correct usage would look like this:

```
x = 1; // initialize x
y = 2; // initialize y
z = x + y; // correct, both x and y are initialized
```

**Error 30. Syntax error**

General syntax error. Occurs when the syntax is incorrect (for example unbalanced parentheses, or unrecognized characters or invalid operators or incorrect/undefined function name or missing brace or semicolon )

```
x = 4;
y = 2;
```

```
z = x * ( 7 + y ; // syntax error here because of missing closing parenthesis
```

### Error 31. Syntax error, expecting <list of tokens>

Occurs The syntax is incorrect because the parser expects specific tokens and finds something else.

For example:

```
while i < 5 // this generates Error 31. Syntax error, expecting '('
```

– the parser expects opening brace '(' after *while* statement

### Error 32. Syntax error, probably missing semicolon at the end of the previous line

General syntax error occurred at the beginning of the line. In most cases it happens because of missing semicolon at the end of the statement in the previous line, see this:

```
a=5
b=4; // <--- here syntax error probably missing semicolon
```

but on some occasions the reason may be simply incorrect syntax at the beginning of the line:

```
a=5;
+b=4; // <-- the same error message but the problem is about an extra + character
at the beginning of the line
```

**Error 33. The identifier is already in use. You attempted to assign value to the identifier representing a function. If you want to return value from function you should use RETURN statement instead.**

Example 1:

```
function Test( x )
{
    return 2 * x;
}

VarSet( "Test", 7 ); // error, identifier 'Test' is already used for function
```

Example 2:

(incorrect)

```
function Test( x )
{
    Test = 2 * x; // error here because Test identifier is already used for
function,
```

```
// you should use return statement to return values from function
}  
  
x = Test(5);
```

Correct function returning value would look like this:

```
function Test( x )  
{  
    return 2 * x; // correct, returning values using return statement  
}  
  
x = Test(5);
```

**Error 34. The identifier is already in use. You attempted to define the function that has the same identifier as global variable.**

Occurs when function definition uses the same identifier as global variable defined earlier in the formula.

```
Test = 5; // global variable  
  
function Test( x ) // incorrect, 'Test' identifier is already used for global  
variable  
{  
    return 2 * x;  
}
```

**Error 35. Shift+BREAK pressed. Loop terminated.**

Occurs when user manually terminates loop execution by pressing Shift+BREAK keys.

**Error 36. N-th argument of the function call has no value set**

May happen if N-th argument of the function is set to no value. This can not happen under normal circumstances, but only when calling functions from inside the plugin or by setting variables from inside plugin.

**Error 37. Unsupported field in SetOptions**

Occurs when wrong (or not supported) field name was used in SetOption call, for example:

```
SetOption("NoSuchOption", 1 );
```

**Error 38. Unsupported field in GetOptions**

Occurs when wrong (or not supported) field name was used in GetOption call, for example:

```
x = GetOption( "NoSuchOption" );
```

**Error 39. CategoryAddSymbol: Setting sector is unsupported, set industry instead**

Occurs when categorySector is used in CategoryAddSymbol function. You can not add symbol to sector because symbols are linked to industry groups only and industry groups are then assigned to sectors. One sector usually includes several industry groups and if you set the industry then sector is implicit. Refer to AmiBroker Users Guide for more information about categories.

```
CategoryAddSymbol( "", categorySector, 2 ); // wrong, you can not use
categorySector
```

```
CategoryAddSymbol( "", categoryIndustry, 2 ); // correct
```

**Error 40. CategoryRemoveSymbol: Removing from sector is unsupported, remove from industry instead**

Occurs when categorySector is used in CategoryRemoveSymbol function. You can not remove symbol from sector because symbols are linked to industry groups only and industry groups are then assigned to sectors. One sector usually includes several industry groups and if you set the industry then sector is implicit. Refer to AmiBroker Users Guide for more information about categories.

```
CategoryRemoveSymbol( "", categorySector, 2 ); // wrong, you can not use
categorySector
```

```
CategoryRemoveSymbol( "", categoryIndustry, 2 ); // correct
```

**Error 41. Unsupported field in GetRTData**

Occurs when not supported field was specified in GetRTData call:

```
GetRTData( "EPSRank" ); // EPSRank is not available from RT sources
```

Note that GetRTData is supported only for real-time data sources and in Professional edition only. If you attempt to call it without running RT data source or using AmiBroker Standard edition, it will quietly return Null value without displaying any error message.

**Error 42. #include failed because the file does not exist: <filename> (current working directory is '...')**

Occurs when specified include file does not exist. Example:

```
#include "not\existing\path\to\the\file.afl"
```

**Error 43. Variable stops are not supported in Rotational Trading mode**

Occurs on attempt to use variable amount in ApplyStop() function when using rotational trading backtester mode. Example:

```
EnableRotationalTrading();

ApplyStop( stopTypeLoss, stopModePoint, H-L ); // variable stop amount not
supported in rotational mode
```

**Error 44. SectorID() is outside 0..63 range.**

May occur during call to SectorID() function if external data plugin sets sector IDs incorrectly (outside 0..63 range)

```
x = SectorID(); // the formula is correct, but may fail with error 44 if data
plugin sets sectors incorrectly
```

**Error 45. Failed to launch trading interface**

Problem occurs when formula calls GetTradingInterface but required interface is not installed or registered properly.

```
ti = GetTradingInterface("DUMMY"); // fails because DUMMY interface is not
installed
```

**Error 46. Missing comma**

Problem occurs when there is a missing comma in the function declaration formal parameter list

```
function MyFun( x y ) // missing comma in the formal parameter list
{
    return x * y;
}
```

**Error 47. Exception occurred during AFL formula execution**

This error occurs when formula caused unhandled system exception. System exception may be due to memory overflow, accessing incorrect file handle, memory access violation, etc.

Example:

```
fclose( 123 ); // closing invalid file handle causes system exception
```

**Error 48. N-volume bar compressed data longer than base time frame. Use higher compression factor.**

This error occurs when N-volume compression setting produces data longer than original data set.

N-volume bars are very different from time-based bars (compression of data to N-volume bar may actually deliver MORE output bars – for example if one tick is 1000 shares and you have specified 100V bars then single tick will be expanded to TEN 100V bars – ten times original size)

TimeFrame functions are protected against array overrun and will not decompress beyond original array size (you will get an "Error 48"). Also switching main time frame to some weird N-volume bar value will result in limiting the output to maximum twice original data size (without error message).

You should keep that in mind and avoid using too small N-volume bar intervals that could lead to such condition. Also due to the nature of N-volume bars the only TimeFrameSet() function will yield correct N-volume bar values, TimeFrameGetPrice() may give slightly distorted results.

Example:

```
TimeFrameMode( 2 );
TimeFrameSet( 20 ); // possible Error 48 - 20-share bar compression may be too
small
```

#### Error 49. Optimization parameter name must not be empty.

This error occurs when Optimize() function is called with empty name argument.

Example:

```
period = Optimize("", 10, 10, 20, 1 ); // WRONG: name must NOT be empty
```

#### Error 50. Optimization parameter minimum value must be less than or equal to maximum and step parameter needs to be greater than zero

This error occurs when AFL's Optimize() function is called with minimum value greater than maximum or step less not greater than zero

Example:

```
period = Optimize("Period", 1, 20, 10, 1 ); // WRONG: minimum > maximum
period2 = Optimize("Period2", 1, 10, 20, 0 ); // WRONG: step = 0
```

## Calculating multiple-security statistics with AddToComposite function

The vast majority of AFL functions operate on single security prices. There are two exceptions from this rule provided by **RelStrength()** and **Foreign()** functions. These two functions allow you to use other security prices in the AFL formula. Although these functions are very useful for things like relative performance charts, they are not so useful for tasks requiring prices of all securities (or a large number of securities) because one would need to type several hundreds of **Foreign()** function calls to do so. Moreover this approach would require listing all the ticker names within the formula which makes the formula tight to particular market. We obviously need completely different approach...

Just imagine if we were able to store the results of calculations performed on single security somewhere and then use those partial data to generate some multiple security indicator. You may say that one can create the exploration, then export the results to the CSV file, then load it into Excel and then perform the calculations there. It would work (in some cases) but you have to agree that the solution is not nice.

This is the area where AddToComposite function can help.

Basically the concept behind AddToComposite is that we run our formula (using Scan feature) through a group of symbols performing some calculations. We will compute some multiple security statistics and store the results in the artificial ticker created using AddToComposite function.

### 2.3 The solution

The key to the solution is the following algorithm:

1. Do some ordinary AFL calculations using any of available functions
2. Add the result of the calculations to one of the O, H, L, C, V, I fields of our artificial ticker (named for example "~composite")

When the above procedure is repeated over a group of symbols our composite ticker will contain the sum of results of individual symbol calculations.

Step 2 described above is implemented entirely inside AddToComposite function:

SYNTAX	AddToComposite( array, "ticker", "field", flags = atcFlagDefaults )
RETURNS	NOTHING
FUNCTION	<p>Allows you to create composite indicators with ease.</p> <p>Parameters:</p> <p>array – the array of values to be added to "field" in "ticker" composite symbol</p> <p>"ticker" – the ticker of composite symbol. It is advised to use ~comp (tilde at the beginning)</p> <p>newly added composites are assigned to group 253 by default and have "use only local database" feature switched on for proper operation with external sources</p> <p>possible field codes: "C" – close , "O" – open, "H" – high, "L" – low, "V" – volume, "I" – open interest, "X" – updates all OHLC fields at once</p> <p>flags – contains the sum of following values</p> <ul style="list-style-type: none"> <li>• atcFlagResetValues = 1 – reset values at the beginning of scan (recommended)</li> <li>• atcFlagCompositeGroup = 2 – put composite ticker into group 253 and EXCLUDE all other tickers from group 253 (avoids adding composite to composite)</li> <li>• atcFlagTimeStamp = 4 – put last scan date/time stamp into FullName field</li> <li>• atcFlagEnableInBacktest = 8 – allow running AddToComposite in backtest/optimization mode</li> <li>• atcFlagEnableInExplore = 16 – allow running AddToComposite in exploration mode</li> <li>• atcFlagResetValues = 32 – reset values at the beginning of scan (not required if you use atcFlagDeleteValues)</li> <li>• atcFlagEnableInPortfolio = 64 – allow running AddToComposite in custom portfolio backtester phase</li> </ul>



- atcFlagDefaults = 7  
(this is a composition of atcFlagResetValues |  
atcFlagCompositeGroup | atcFlagTimeStamp flags)

AddToComposite function also detects the context in which it is run (it works ONLY in scan mode, unless atcFlagEnableInBacktest or atcFlagEnableInExplore flags are specified) and does NOT affect composite ticker when run in Indicator or Commentary mode, so it is now allowed to join scan and indicator into single formula.

EXAMPLE      AddToComposite( MACD() > 0, "~BullMACD", "V");  
graph0 = Foreign("~BullMACD", "V");

(now you can use the same formula in scan and indicator)

AddToComposite function opens up a huge variety of interesting applications. The following examples will help you understand what you can do with AddToComposite function.

#### Example 1:

Let's say we want to create custom index (average of prices of multiple tickers). With AddToComposite function you can do this fairly easy:

```
/* AddToComposite statements are for Automatic Analysis -> Scan */
/* add Close price to our index OHLC fields */
AddToComposite(Close, "~MyIndex", "X" );

/* add one to open interest field (we use this field as a counter) */
AddToComposite( 1, "~MyIndex", "I" );

buy = 0; // required by scan mode

/* this part is for Indicator */
graph0 = Foreign( "~MyIndex", "C" )/Foreign( "~MyIndex", "I" );
```

You should use above the formula in Automatic Analysis -> Scan mode (over the group of symbols of your choice). This will create "~MyIndex" artificial ticker that will contain your index.

Shortly this formula just adds Close price to OHLC fields (the "X" field stands for all OHLC) of our artificial ticker ~MyIndex. Additionally we add "1" to "I" (open interest) field – effectively counting the number of symbols scanned. We can use symbol count later on to divide the sum of prices by the number of symbols included ( the last line of the formula above ).

#### Example 2:

In the second example we will show how to calculate the indicator that shows the number of symbols meeting certain criterion. In this example this would be RSI less than 30 (oversold condition), but it can be anything you like.

So the first line of our formula will be:

```
values = rsi() < 30;
```

This will store "true" in the values array for all date points when RSI is less than 30. Then we add regular AddToComposite part:

```
buy = 0; // do not generate signals
AddToComposite( values, "~MyComposite", "V" );
```

If we run the formula using "Scan" function of Automatic Analysis window the result would be an artificial symbol "~MyComposite" filled with quotations. The Volume field of those quotes will contain the number of symbols meeting our criterion (RSI<30) in the population of scanned symbols.

You can easily see the chart of this new "indicator" using the following custom formula:

```
graph0 = foreign( "~MyComposite", "V" );
```

High values of this "indicator" show that most of the symbols in the analysed group are oversold. This usually happens before a great rise of the whole market. We just created market-wide oversold detector!

### Example 3:

In the third example I will show you how to use the same technique to count the number of open positions of your trading system. This is useful if you want to know how big account would you need to trade your system following all the trades. Our formula will be very similar to the one before.

First we should have our original trading system formula:

```
/* Your original formula here */
/* In this example this is simple macd/signal crossover system)

buy = cross( macd(), signal() );
sell = cross( signal(), macd() );

/* the following line uses Flip function to get "1" after the buy
signal and reset it back to "0" after sell appears. */

in_trade = flip( buy, sell );

AddToComposite( in_trade, "~OpenPosCount", "V" );
```

We use "~OpenPosCount" artificial ticker to store the results. Again we should run just Scan of the formula and the "~OpenPosCount" ticker would become available.

Use

```
graph0 = foreign( "~OpenPosCount", "V" );
```

after running the back-test to see the chart of the number of open positions of your system.

## 2.4 Notes

For more details on composites check "[Introduction to AddToComposite](#)" (122KB PDF) by Herman van den Bergen.

Please note that to update any composite ticker (for example after adding/editing quotes) you should run "Scan" again.

The idea was originally presented in the [12/2001 issue of AmiBroker tips newsletter](#). Special thanks to Mr. Dimitris Tsokakis for very constructive discussions that allowed creation and enhancements of this idea.

## Equity function, Individual and Portfolio Equity Charts

### Introduction

The equity line is probably the best diagnostic tool for trading system developer. In one graph it shows the sum total of the success or failure of the system being tested, and the resulting effect on your equity.

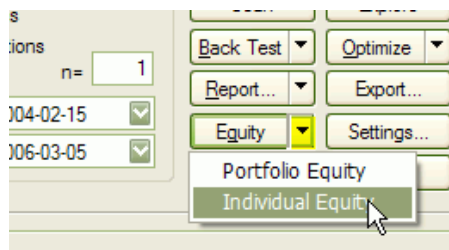
Numbers from [Report](#) telling of a drawdown is nice but with a graph, one can see how things were going before, during, and after a drawdown.

The line produced by the equity function tracks the amount of equity in your account. So, for example, if you backtest a system over the last year, you should see that, at the beginning of that year, the line starts at the amount of your initial equity, and then rises and falls because, as each trade makes or loses money, the equity in your account will rise and fall. It shows how much money is in your account throughout the backtest period, so you can visually see how your system performed.

So, clearly you don't want to see the line go down – that means you lost money. It is generally accepted that you want to revise your system test parameters in order to get as close as possible to a smooth, straight, rising line. This means that your system has performed consistently over time, and presumably over different market conditions. A line that goes up and down frequently means that your system works well only under certain conditions, and poorly under other conditions.

### Individual (single-security) Equity chart

To display single security Equity chart it is enough to click on the drop down arrow on the "Equity" button and choose "Individual Equity" from the menu in the [Automatic Analysis window](#) AFTER running a backtest.



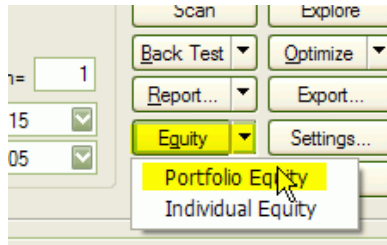
This will plot the equity for currently active symbol and recently backtested system. If you want to see the Equity curve for another symbol – just switch to this symbol and Equity line will be recalculated automatically.

You can also choose symbol that was not included in the original backtest set and AmiBroker will calculate correct equity curve as it would look if real backtest was performed on it.

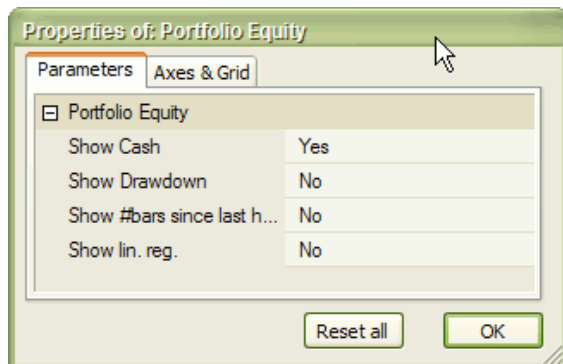
**IMPORTANT:** individual equity chart is single–security equity that does not show portfolio–level effects like skipping some of trades due to reaching maximum open position limit or funds being allocated to other securities, it also does not use some advanced functionality offered only by portfolio–level backtester. For more information [see this](#).

### Portfolio–level Equity chart

To display portfolio–level equity chart is is enough to double click on "Equity" button in the Automatic Analysis window or click on the drop down arrow on the "Equity" button and choose "Portfolio Equity" from the menu **AFTER** running a backtest.



Portfolio–level equity represents equity of your entire portfolio and reflects ALL real–world effects like skipping trades due to insufficient funds, reaching maximum number of open positions. It also reflects all scaling in/out, HoldMinBars effect, early exit fees and any other feature that you may have been using in your formula. Portfolio–level equity also by default shows the remaining cash in the portfolio. Using [Parameters window](#) (click with RIGHT mouse button over equity chart and select "Parameters" from the menu) you can turn on display of drawdown (underwater equity curve), number of bars since last equity high and linear regression of the equity.



### Equity function

Equity() function is a single–security backtester–in–a–box. It has many interesting applications that will be outlined here. Let's look at the definition of Equity function:

<b>SYNTAX</b>	<b>equity</b> ( <i>Flags</i> = 0, <i>RangeType</i> = -1, <i>From</i> = 0, <i>To</i> = 0 )
<b>RETURNS</b>	ARRAY
<b>FUNCTION</b>	Returns Equity line based on buy/sell/short/cover rules, buy/sell/short/coverprice arrays, all apply stops, and all other backtester settings.
	<i>Flags</i> – defines the behaviour of Equity function

**0** : (default) Equity works as in 3.98 – just calculates the equity array  
**1** : works as 0 but additionally updates buy/sell/short/cover arrays so all redundant signals are removed exactly as it is done internally by the backtester plus all exits by stops are applied so it is now possible to visualise ApplyStop() stops.  
**2** : (advanced) works as 1 but updated signals are not moved back to their original positions if buy/sell/short/cover delays set in preferences are non-zero. Note: this value of flag is documented but in 99% of cases should not be used in your formula. Other values are reserved for the future.

*RangeType* – defines quotations range being used:

**-1** : (default) use range set in the Automatic analysis window  
**0** : all quotes  
**1** : n last quotes (n defined by 'From' parameter)  
**2** : n last days (n defined by 'From' parameter)  
**3** : From/To dates

*From* : defines start date (datenum) (when RangeType == 3) or "n" parameter (when RangeType == 1 or 2)

*To*: defines end date (datenum) (when RangeType == 3) otherwise ignored

datenum defines date the same way as DateNum() function as YYYYMMDD where YYYY is (year – 1900), MM is month, DD is day

December 31st, 1999 has a datenum of 991231  
 May 21st, 2001 has a datenum of 1010521

All these parameters are evaluated at the time of the call of Equity function. Complete equity array is generated at once. Changes to buy/sell/short/cover rules made after the call have no effect. Equity function can be called multiple times in single formula.

#### EXAMPLE

```
buy = your buy rule;
sell = your sell rule;
graph0 = Equity();
```

#### SEE ALSO

Using Equity function we can build up Equity "indicator" that will work without the need to run backtester. Just type the following formula in the Formula Editor and press Apply:

```
buy = ... your buy rule ...
sell = .... your sell rule ...

graph0 = Equity();
```

Equity() function uses the buy/sell/short/cover rules that are defined BEFORE this function is called. The whole backtesting procedure is done inside Equity function that generates equity line.

## Notes:

1. Equity line is dependant of the parameters in the Automatic Analysis settings
2. Equity traces Interest Earnings when you are OUT of the market.  
If you don't want this just enter 0 into "Annual interest rate" field in the settings.
3. Equity also traces commissions. If commissions are not zero entry commission is taken using position size of the entry and exit commission is taken for each point to simulate how much money would you have if you closed position at given bar.
4. AmiBroker uses SellPrice array for long trades and CoverPrice array for short trades to calculate current equity value.
5. Equity() function is single–security and does not reflect portfolio–level effects like skipping trades, does not handle some advanced functionality offered only by portfolio–backtester. For more information see [that table](#).

**Portfolio Equity special symbol**

After running portfolio–level backtest, AmiBroker writes the values of portfolio equity to special symbol "~~~EQUITY". This allows you to access portfolio–level equity of last backtest inside your formula. To do so, use Foreign function:

```
PortEquity = Foreign( "~~~EQUITY", "C" );
```

This is exactly what for example built–in portfolio equity chart formula is doing.

**Can I calculate system statistics using Equity function?**

Yes you can. Here is a couple of example calculations kindly provided by Herman van den Bergen:

```
E = Equity();

//How rich you were :-)
Plot(Highest(E,"I should have sold",1,3);

//Your Current Drawdown:
Plot(Highest(E) - E,"Current DrawDown",4,1);

//Trade profits:
LongProfit = IIf(Sell,E - ValueWhen(Buy,E),0);
ShortProfit = IIf(Cover,ValueWhen(Short,E)-E,0);
Plot(IIf(Sell,LongProfit,0),"LProfit",8,2+4);
Plot(IIf(Cover,ShortProfit,0),"SProfit",4,2+4);

//Current Trade Profit:
Lastbar = Cum(1) == LastValue( Cum(1) );
Plot(IIf(LastBar AND pos,E-ValueWhen(Buy,E),ValueWhen(Short,E) -
E),"Current Profit",9,2+4);

//DailyProfits:
Plot(IIf(pos,E-Ref(E,-1),Ref(E,-1)-E),"Daily Profits",7,2);
```

**How do you plot a composite Equity Curve ? I don't want the whole database, but would like to see the curve based on the watchlist I use for the backtesting.**

Just use Portfolio Level equity chart (see above). It represents actual portfolio backtest equity, so if you run your backtest on the watch list it will represent what you need. You can also write your own formula that does the same thing:

```
Plot( Foreign( "~~~EQUITY", "C" ), "PL Equity", colorRed );
```

## Functions accepting variable periods

The following functions support variable periods (where periods parameter can be array and change from bar to bar):

- AMA
- AMA2
- DEMA
- HHV
- HHVBars
- LinRegSlope
- LinearReg
- LinRegIntercept
- LLV
- LLVBars
- MA
- Ref
- StdErr
- Sum
- TEMA
- TSF
- WMA

## User-definable functions, procedures. Local/global scope

User-definable functions allow to encapsulate user code into easy-to-use modules that can be user in many places without need to copy the same code over and over again.

Functions must have a definition. The function definition includes the function body the code that executes when the function is called.

A function definition establishes the name, and attributes (or parameters) of a function. A function definition must precede the call to the function. The definition starts with **function** keyword then follows function name, opening parenthesis then optional list of arguments and closing parenthesis. Later comes function body enclosed in curly braces.

A function call passes execution control from the calling function to the called function. The arguments, if any, are passed by value to the called function. Execution of a return statement in the called function returns control and possibly a value to the calling function.

If the function does not consist of any return statement (does not return anything) then we call it a procedure.

Following is an example of function definition:

```
// the following function is 2nd order smoother

function IIR2( input, f0, f1, f2 )
{
    result[ 0 ] = input[ 0 ];
    result[ 1 ] = input[ 1 ];

    for( i = 2; i < BarCount; i++ )
    {
        result[ i ] = f0 * input[ i ] +
                     f1 * result[ i - 1 ] +
                     f2 * result[ i - 2 ];
    }

    return result;
}

Plot( Close, "Price", colorBlack, styleCandle );
Plot( IIR2( Close, 0.2, 1.4, -0.6 ), "function example", colorRed );
```

In this code **IIR2** is a user-defined function. **input**, **f0**, **f1**, **f2** are formal parameters of the functions.

At the time of function call the values of arguments are passed in these variables. Formal parameters behave like local variables.

Later we have **result** and **i** which are local variables. Local variables are visible inside function only. If any other function uses the same variable name they won't interfere between each other.

Due to the fact that AFL does not require to declare variables the decision whenever given variable is treated as local or global is taken depends on where it is FIRST USED.



If given identifier appears first **INSIDE** function definition – then it is treated as **LOCAL** variable.

If given identifier appears first **OUTSIDE** function definition – then it is treated as **GLOBAL** variable.

This default behaviour can be however overridden using **global** and **local** keywords (introduced in 4.36) – see example 2.

*Example (commentary):*

```
k = 4; // this is GLOBAL variable

function f( x )
{
    z = 3; // this is LOCAL variable
    return z * x * k; // 'k' here references global variable k (first used above
    outside function)
}

z = 5; // this is GLOBAL variable with the same name as local variable in
function f

"The value of z before function call :" + WriteVal( z );

// Now even if we call function
// the value of our global variable z
// is not affected by function call because
// global variable z and local variable z are separate and
// arguments are passed by value (not by reference)

"The result of f( z ) = " + WriteVal( f( z ) );

"The value of z after function call is unchanged : " + WriteVal( z );
```

*Example 2: Using local and global keywords to override default visibility rules:*

```
VariableA = 5; // implicit global variable

function Test()
{
    local VariableA; // explicit local variable with the same identifier as
    global          // global
    global VariableB; // explicit global variable not defined earlier
                    // may be used to return more than one value from the
    function

    VariableA = 99;
    VariableB = 333;
}

VariableB = 1; // global variable

"Before function call";
"VariableA = " + VariableA;
```

```

"VariableB = " + VariableB;

Test();

"After function call";
"VariableA = " + VariableA + " (not affected by function call)";
"VariableB = " + VariableB + " (affected by the function call)"

```

At the end of the function we can see 'return' statement that is used to return the result to the caller. Note that currently return statement must be placed at the very end of the function.

It is also possible to write a procedure (a function that returns nothing (void))

```

procedure SinePlotter( Freq, ColorIndex )
{
    pname = "Line"+WriteVal(ColorIndex,1.0);
    array = sin( Cum( Freq * 0.01 ) );
    Plot( array, pname , colorRed + ColorIndex, styleThick );
}

for( n = 1; n < 10; n++ )
{
    SinePlotter( n/2+Cum(0.01), n );
}

```

Note that although there are two separate keywords 'function' and 'procedure' AmiBroker currently treats them the same (they both accept return values but not require them), but in the future the rules might get enforced to use

return statement ONLY in conjunction with function keyword. So it is advised to use function keyword in case when your function returns any value and procedure keyword otherwise.

Note also that recursion (having a function call itself from within itself) is NOT supported as for now.

### More information

Please read also [Understanding how AFL works](#) article to learn more.

## AFL Tools

### Automatic technical analysis

#### Introduction

Since version 2.5 AmiBroker features automatic technical analysis tools. AmiBroker can check for user defined buy/sell conditions giving you an idea about the current situation on the market. It can also perform a system test (simulation) telling you about the performance of your trading system. Version 3.0 of AmiBroker introduced new formula language (AFL) allowing you to write not only system tests but also custom indicators and guru advisor commentaries.

In order to do this you have to define buy and sell rules, indicator formulas or commentaries using a special *AmiBroker Formula Language (AFL)*, which is described below. For more information about using analysis tools see also the description of the *Automatic analysis window*, *Formula Editor*, and *Commentary window* in Chapter 2.

## AmiBroker Formula Language

AFL is used for defining your trading rules and explorations in Automatic analysis window, custom commentaries in the Guru Commentary window and indicator formulas in [Formula Editor](#) window.

Detailed reference of AFL language is given [here](#).

## Examples

Below you will find some simple buy and sell rules. They are just formal equivalents of some of the most common indicators' interpretation rules. You can treat them as a starting point for developing your own trading strategies, but before you get too excited when you think you've found the "holy grail" of trading systems, check the following:

- Test the system on different symbols and different time frames. The results should be similar to those on the original data tested.
- Test the system on different types of markets (e.g., upward trending, downward trending, and sideways). A good system should work in all types of markets, since you won't always know when a market changes from trending to trading or vice versa.
- Pay close attention to the number of trades generated by a trading simulation. If there are a large number of trades and large profits, be sure you specified realistic commissions. The results of the test may be much different once commissions are factored in.

```
buy = cross( macd(), 0 );
sell = cross( 0, macd() );
```

```
buy = cross( ema( close, 9 ), ema( close, 15 ) );
sell = cross( ema( close, 15 ), ema( close, 9 ) );
```

```
buy = cross( rsi(), 30 );
sell = cross( 70, rsi() );
```

```
buy = cross( ultimate(), 50 );
sell = cross( 50, ultimate() );
```

## Automatic analysis window

Automatic analysis window enables you to check your quotations against defined buy/sell rules. AmiBroker can produce report telling you if buy/sell signals occurred on given symbol in the specified period of time. It can also simulate trading, giving you an idea about performance of your system.

In the upper part of window you can see text entry field. In this field you should enter buy and sell rules. These rules are assignment statements written in AmiBroker's own language. You can find the description of this language in [AFL reference guide](#).

In order to make things work you should write two assignment statements (one for buy rule, second for the

sell rule), for example:

```
buy = cross( macd(), 0 );  
sell = cross( 0, macd() );
```

Automatic analysis window allows you also to optimize your trading system and perform in-depth explorations

See also: [detailed description of Automatic Analysis window controls](#)

## Formula Editor

Formula Editor allows you to write formulas to be used as indicators or in Automatic Analysis window. More on this [here](#).

## Guru Advisor Commentary window

Commentary window enables you to view textual descriptions of actual technical situation on given market. Commentaries are generated using formulas written in AmiBroker's own formula language. You can find the description of this language in [AmiBroker Formula Language Reference Guide](#).

Moreover Commentary feature gives you also graphical representation of buy & sell signals by placing the marks (arrows) on the price chart.

NOTE: Using complex commentary formulas you may observe long execution times.

See also: [detailed description of Guru Advisor Commentary window controls](#)

## AFL Scripting Host

**IMPORTANT NOTE:** Since the introduction of native looping and flow control statements like *if-else* and *while* in version 4.40 the significance of scripting has been greatly reduced. Currently most of the tasks requiring scripting in previous versions could be handled in native AFL. What's more AFL loops are 3–6 times faster than JScript/VBScript.

### Basics

AFL scripting host is an interface between AFL engine and JScript/VBScript engines available as a part of Internet Tools & Technologies platform provided by Microsoft. It allows you to build the formulas that have parts in AFL code and parts in JScript/VBScript.

### Requirements

- AmiBroker 3.59 or higher
- Microsoft JScript/VBScript engines installed

Microsoft JScript/VBScript engines come with Internet Explorer 4 or higher (Windows 98, Millenium, 2000 have it included in the operating system). It is however advised to install the latest version of Internet Explorer (5.5) or download and install the latest version of Windows Scripting Host (5.5) from Microsoft:

Windows 95, 98, Me, NT:

<http://www.microsoft.com/scripting/downloads/v55/other/scr55en.exe>

Windows 2000:

<http://www.microsoft.com/scripting/downloads/v55/windows2000/scripten.exe>

JScript/VBScript documentation can be found on official scripting page at:

<http://msdn.microsoft.com/scripting/>

## Enabling AFL Scripting Host

If you want to use scripts within your formulas you have to call EnableScript() function in the beginning of your formula. The function takes one input parameter – engine name:

```
EnableScript("jscript");
```

or

```
EnableScript("vbscript");
```

From then on, you will be able to embody parts written in scripting language in your formulas. The begin and the end of the script must be marked with <% and %> sequences, as shown in the example below:

```
EnableScript("vbscript");

// "normal" AFL statements
buy = cross( macd(), 0 );
sell = cross( 0, macd() );

<%
..... your script code here.....
%>

// "normal" AFL statements
buy = ExRem( buy, sell );
```

## Using variables

Currently the only way to exchange the information between "normal" AFL part and script part is to use variables. AFL scripting host exposes one object (predefined, no creation/initialization needed) called **AFL**.

The AFL object has one (default) parametrized property called Var( varname ) that can be used to access AFL variables from the script side:

```
// example in VBScript
<%

buyarrayfromscript = AFL.Var("buy")
    ' get the buy array from AFL to the script-defined variable

AFL.Var("buy") = buyarrayfromscript
    ' set the buy array in AFL from the script-defined variable

%>
```

Since Var is default property you can omit its name and write simply AFL( varname ) as shown in the example below:

```
<%

buyarrayfromscript = AFL("buy")
    ' gets the buy array from AFL to the script-defined variable

AFL("buy") = buyarrayfromscript
    ' sets the buy array in AFL from the script-defined variable

%>
```

In AFL there are three data types possible: array (of floating point numbers), a number (floating point) and a string. The VBScript and JScript engines use variant data type that can hold any type of the variable including three used by AFL. As in AFL, you don't declare variables in scripting languages, the type is determined by the first assignment. In case of VBScript you can get/set AFL variables of any supported type using syntax shown above. But in JScript, due to the fundamental difference in handling arrays in JScript (array elements in JScript are implemented as dynamic properties of an array object) you need to use the following code to get the value of AFL array into JScript array:

```
// example in JScript
<%
function GetArray( name )
{
    return VBAArray( AFL( name ) ).toArray();
}

myJScriptArray = GetArray( "close" );
%>
```

The GetArray() function shown above makes it easy to convert automation-type safe array into JScript array. This example shows also how to define and use functions in JScript;

Assigning AFL variables from script-side arrays is much more simple, AFL scripting host detects JScript arrays and can get their contents directly:

```
// example in JScript
<%
    AFL("buy") = myJScriptBuyArray;
%>
```

All other data types are handled the same in JScript and VBScript

```
// example in VBScript
ticker = name();
<%
    tickerstring = AFL("ticker")
    AFL("ticker") = "new name"
%>
```

or in JScript:

```
// example in JScript
ticker = name();
<%
    tickerstring = AFL("ticker");
    AFL("ticker") = "new name";
%>
```

## Iterating through arrays

One of the most basic task that everyone would probably do is to iterate through array. In VBScript this can be done using For..To..Next statement, in JScript using for(;;) statement. Both these constructs need to know array size or number of elements in the array. In VBScript you should use UBound( array) function to get the upper bound of the array, in JScript you just use length property of the array. The following examples show this. (Please remember that in both VBScript and JScript arrays are zero-based.)

```
// example in VBScript
<%
myArray = AFL("close")

sum = 0
for i = 0 to UBound( myArray )

    sum = sum + myArray( i )

next

%>
```

or in JScript:

```
// example in JScript
<%
function GetArray( name )
{
    return VBAArray( AFL( name ) ).toArray();
}

myArray = GetArray( "close" );

sum = 0;

for( i = 0; i < myArray.length; i++ )
{
    sum += myArray[ i ];
}

%>
```

**Examples****a) Indicator example – Exponential moving average:**

```

EnableScript("jscript");
<%

close = VBAArray( AFL( "close" ) ).toArray();

output = new Array();

// initialize first element
output[ 0 ] = close[ 0 ];

// perform the loop that calculates exponential moving average
factor = 0.05;

for( i = 1; i < close.length; i++ )
{
    output[ i ] = factor * close[ i ] + (1 - factor) * output[ i - 1 ];
}

AFL.Var("graph0") = close;
AFL.Var("graph1") = output;

%>
WriteVal( graph1 );

```

**b) Profit–target stop example**

Here comes the example of the formula that realizes profit–target stop at the fixed 10% percentage from the buy price. Note that buy condition is met when the price reaches a new high, so it happens multiple times after initial buy. Therefore ValueWhen( buy, close ) can not give you initial buy price and that kind of trading rule could not be implemented in AFL itself. But, with scripting there is no problem...

```

EnableScript("VBScript");

hh = HHV( close, 250 );

// buy when prices reaches a new high
buy = Close == HHV( close, 250 );

// ensure that sell is an array,
// sell = 0 would set the type to number
sell = buy;

<%
    close = AFL("close")
    buy = AFL( "buy" )
    sell = AFL("sell")

    ' this variable holds last buying price
    ' if it is zero it means that no trade is open
    lastbuyprice = 0

```



```

        ' iterate along the array
    for i = 0 to UBound( close )
        sell( i ) = 0

        ' Remove Buy signals if trade was already initiated
        if( lastbuyprice > 0 ) then
            buy( i ) = 0
        end if

        ' if there is no open trade and buy signal occurs
        ' get the buying price
        if ( lastbuyprice = 0 ) AND ( buy( i ) = 1 ) then
            lastbuyprice = close( i )
        end if

        ' if trade is open and sell condition is valid
        ' generate sell signal
        ' and close the trade
        if (lastbuyprice > 0 ) AND ( close( i ) > ( 1.1 * lastbuyprice ) ) then
            sell( i ) = 1
            lastbuyprice = 0
        end if
    next

    AFL("buy") = buy
    AFL("sell") = sell

    %>

    buy = buy;

```

## Further information

More scripting samples are available at the AFL on-line library at:  
<http://www.amibroker.com/library/list.php>

In case of any further questions, comments and suggestions please contact me at:  
[support@amibroker.com](mailto:support@amibroker.com) . Please note that AFL scripting is fairly advanced topic and you should play a little bit with AFL first before going too deep into scripting.

## Component Object Model support in AFL

### Introduction

The Component Object Model (COM) is the technology that defines and implements mechanisms that enable software components, such as applications, data objects, controls, and services, to interact as objects. The COM support in AFL introduced in version 3.75beta allows to create instances of COM objects and call the functions (methods) exposed by those objects.

The COM object can be created in virtually any language including any C/C++ flavour, Visual Basic, Delphi, etc. This enables you to write parts of your indicators, systems, explorations and commentaries in the language of your choice and run them at full compiled code speed.

The scripting engines used by AFL scripting host (JScript and VBScript) also expose themselves as COM objects. AFL COM support now allows you to call functions defined in scripting part directly from AFL without

the use of variables to pass the data to and retrieve data from the script.

### ***Calling functions defined in script***

Until version 3.75 the only way to exchange information between AFL and the script was using variables – this technique is explained in detail in [AFL scripting host documentation](#) .

Let's suppose we need a function that calculates second order IIR (infinite impulse response) filter:

$$y[n] = f_0 * x[n] + f_1 * y[n - 1] + f_2 * y[n - 2]$$

Please note that well known exponential smoothing is a first order IIR filter. Implementing higher order filters minimizes lag, therefore our second order IIR may be used as a "better" EMA.

In the "old way" we would need to write the following code:

```
EnableScript("jscript");

x = ( High + Low )/2;

f0 = 0.2;
f1 = 1.2;
f2 = -0.4;

<%
x = VBAArray( AFL( "x" ) ).toArray();
f0 = AFL( "f0" );
f1 = AFL( "f1" );
f2 = AFL( "f2" );

y = new Array();

// initialize first 2 elements of result array
y[ 0 ] = x[ 0 ];
y[ 1 ] = x[ 1 ]

for( i = 2; i < x.length; i++ )
{
    y[ i ] = f0 * x[ i ] + f1 * y[ i - 1 ] + f2 * y[ i - 2 ];
}

AFL.Var("y") = y;
%>

Graph0 = Close;
Graph0Style = 64;
Graph1 = y;
```

While it is OK for one-time use, if we need such a function multiple times we had to have repeat the script part which is not very nice. Much nicer approach is to have a function that can be called from multiple places without the need to repeat the same code. Defining functions in JScript or VBScript is no problem at all:

```

EnableScript("jscript");

<%

function IIR2( x, f0, f1, f2 )
{

    x = VBAArray( x ).toArray();

    y = new Array();

    // initialize first 2 elements of result array
    y[ 0 ] = x[ 0 ];
    y[ 1 ] = x[ 1 ];

    for( i = 2; i < x.length; i++ )
    {
        y[ i ] = f0 * x[ i ] + f1 * y[ i - 1 ] + f2 * y[ i - 2 ];
    }

    return y;

}

%>

```

.. but how to call such a function from AFL?

The most important thing is that script engine exposes itself as a COM object. A new AFL function `GetScriptObject()` can be used to obtain the access to the script engine. The rest is simple – once we define the function in the script it is exposed as a method of script object retrieved by `GetScriptObject()`:

```

script = GetScriptObject();
Graph0 = script.IIR2( ( High + Low )/2, 0.2, 1.2, -0.4 );
Graph1 = script.IIR2( ( Open + Close )/2, 0.2, 1.0, -0.2 ); // call it again and
again...

```

Note also, that with this approach we may pass additional arguments so our IIR2 filter may be re-used with various smoothing parameters.

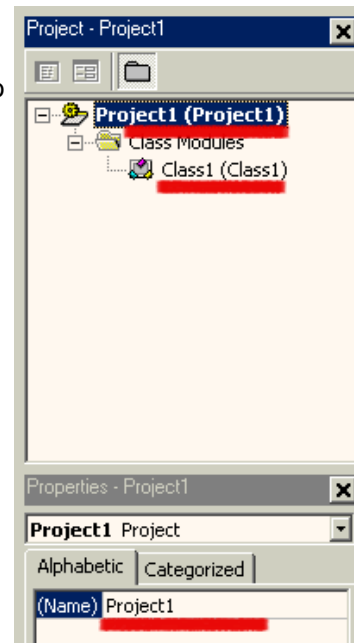
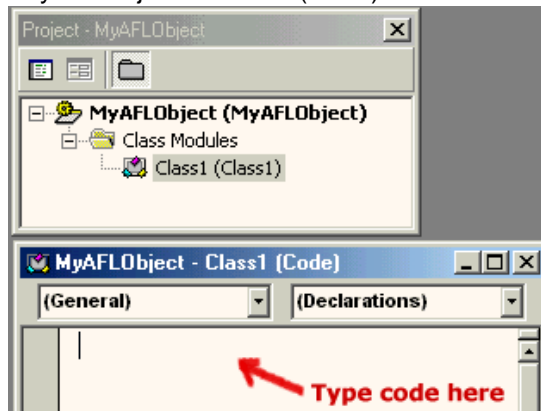
So, thanks to a new COM support in AFL, you can define functions in scripts and call those functions from multiple places in your formula with ease.

### ***Using external COM/ActiveX objects in AFL***

In a very similar way we can call functions (methods) in an external COM objects directly from the AFL formula. Here I will show how to write such external ActiveX in Visual Basic but you can use any other language for this (Delphi for example is very good choice for creating ActiveX/COM objects).

It is quite easy to create your own ActiveX DLL in Visual Basic, here are the steps required:

- Run Visual Basic
- In the "New project" dialog choose "ActiveX DLL" icon
  - this will create the "Project1" that looks like in the picture on the right:
- Now click on the (Name) and rename the "Project1" to something more meaningful, for example "MyAFLObjct"
- Then double click on the "Class1" in the project tree item. The code window will get the title of "MyAFLObjct – Class1 (Code)" as shown below:



- Now you are ready to enter the code

As an example we will implement a similar function to one shown in the JScript. The function will calculate second order Infinite Impulse Response filter. We will call this function "IIR2"

```
Public Function IIR2(InputArray() As Variant, f0 As Variant, f1 As Variant, f2 As Variant) As Variant
```

```
Dim Result()
```

```
ReDim Result(UBound(InputArray)) ' size the Result array to match InputArray
```

```
'initialize first two elements
```

```
Result(0) = InputArray(0)
```

```
Result(1) = InputArray(1)
```

```
For i = 2 To UBound(InputArray)
```

```
    Result(i) = f0 * InputArray(i) + f1 * Result(i - 1) + f2 * Result(i - 2)
```

```
Next
```

```
IIR2 = Result
```

```
End Function
```

The code is quite similar to the JScript version. The main difference is declaring types. As you can see all variables passed from and to AFL must be declared as Variants. This is so, because AmiBroker does not know what kind of object it speaks to and puts all arguments to the most universal Variant type and expects

the function to return the value as Variant also. Currently AmiBroker can pass to your object floating point numbers, arrays of floating point numbers, strings, and pointers to other objects (dispatch pointers) – all of them packed into Variant type. When you write the ActiveX, it is your responsibility to interpret Variants received from AmiBroker correctly.

Now you should choose Run→Start in Visual Basic to compile and run the component. The code in Visual Basic will wait until external process accesses the code.

To access the freshly created ActiveX we will use the following AFL formula (enter it in the Formula Editor and press Apply):

```
myobj = CreateObject("MyAFLObject.Class1");

Graph0 = Close;
Graph0Style = 64;
Graph1 = myobj.IIR2( Close, 0.2, 1.2, -0.4 );
```

The AFL formula simply creates the instance of our ActiveX object and calls its member function (IIR2). Note that we are using new dot (.) operator to access myobj members.

Now click the "Apply" button in the Formula Editor to see how all this setup works. You should see candlestick chart with a quite nice moving average.

## 2.4 Conclusion

Introduction of COM support in AFL brings even more power to AFL and AmiBroker. Now you can write indicators, trading systems, explorations and commentaries using custom functions that are easy to create using scripting language or full-featured development environment of Visual Basic, Borland Delphi, C++ Builder, Visual C++ and many, many others. Using integrated development environments like those mentioned makes debugging, testing and developing much easier and faster. Also resulting compiled code executes several times faster than interpreted script or AFL.

But this is not the end of the story... C/C++ programmers can choose to write plugin DLLs that do not use COM technology at all. Plugin DLLs has some additional features including ability to call back AFL built-in functions, directly retrieve and set AFL variables and support automatic syntax colouring of functions exposed by the plugin. This topic is covered in the AmiBroker Development Kit available from the member's area of AmiBroker site.

## Plug-in in AFL

This section describes regular plug-in DLLs. If you are interested in ActiveX plugins check ["COM support in AFL"](#) section.

### Plugin interface

AmiBroker Plugin interface allows to call an external module (DLL library) directly from AFL. Such a library can expose multiple functions that can be used in your trading systems, indicators, commentaries, scans and explorations. The plugin DLL can be created using any C/C++ compiler, Delphi and other languages supporting creation of regular DLLs. As the code of a plug in is compiled to a native processor machine code

it runs several times faster than AFL. Also since you can use full power of C/C++ (or other) language to build the most complex functions with ease.

In addition to exposing functions to AFL, plugin DLLs have ability to call back AFL built-in functions, directly retrieve and set AFL variables and support automatic syntax colouring of functions exposed by the plugin.

A detailed description of plugin interface and the sample code of plugin DLL are included in the AmiBroker Development Kit (ADK). **The ADK is available for registered users only (downloadable from members area).**

### Getting 3rd party plugins

Freeware 3rd party plugins are available for download from <http://www.amibroker.net/3rdparty.php>

### Using third-party plugins:

To use third-party plugin DLL just copy the DLL file to the Plugins folder in the AmiBroker directory. Then run AmiBroker. Then choose Tools->Plugins menu. In the [Plugins window](#) you should see the list of all loaded plugin DLLs. If AmiBroker was running when you copied the DLL you should click on "Unload" and then on "Load" button. This will force rescanning the Plugins folder and loading the DLLs.

When the plugin DLL is loaded the new functions exposed by this DLL become available to all your AFL formulas. For the list of functions exposed by plugin you should consult the documentation of the plugin itself.

**IMPORTANT NOTE: AmiBroker makes no representations on features and performance of non-certified third-party plug-ins. Specifically certain plug-ins can cause instabilities or even crashes. Entire use of non-certified third-party plugins is at your own risk.**

## Common Coding mistakes in AFL

This document presents most common mistakes and problems that users encounter when writing their custom formulas. Please read carefully to avoid making similar errors.

- “=” (assignment) vs “==” (equality check)
- Using parentheses
- If function
- If is for arrays, Writelf is for strings
- if-else statement needs boolean (or single numeric expression), not array
- Barcount vs BarIndex()
- TimeFrameExpand( ) is required to match data with original time frame

---

### “=” (assignment) vs “==” (equality check)

There are two similar looking but completely different operators in AFL.

“=” is a variable assignment operator

"==" is an equality check operator

### EXAMPLE

*Incorrect code:*

```
result = IIf( Variable = 10 , High, Low ); // WRONG
```

If you want to check if variable is equal to 10, you MUST use ==

*Correct code:*

```
result = IIf( Variable == 10 , High, Low ); // CORRECT
```

---

### Using parentheses

Parentheses can be used to control the operation precedence (the order in which the operators are calculated). AmiBroker always does operations within the innermost parentheses first. To learn the the precedence of operations when parentheses are not used, visit:

[http://www.amibroker.com/guide/a\\_language.html](http://www.amibroker.com/guide/a_language.html)

### EXAMPLE:

"I would like to buy whenever either Close is higher than it's 10-periods Moving Average or Close is at least 10% higher than yesterday's close, but buy should only apply when Current Volume is higher than it's 10-period Moving Average. However – I get Buy signals for the days when Volume is lower than MA(Volume,10). Why?"

```
Buy = Close > MA( Close, 10 ) OR Close == 1.1 * Ref( Close, -1 ) AND Volume > MA(
Volume, 10 );
```

The solution is to add parentheses, otherwise system buys whenever **Close > MA(Close,10)** condition is met ( or **Close == 1.1\*Ref(Close,-1) AND Volume > MA(Volume,10)** are both met).

```
Buy = ( Close > MA( Close, 10 ) OR Close == 1.1 * Ref( Close, -1 ) )
      AND Volume > MA( Volume, 10 );
```

---

### IIf function

The IIf( ) function is used to create **conditional assignments**.

```
variable = IIf( EXPRESSION, TRUE_PART, FALSE_PART );
```

The above "IIf" statement means: For each bar EXPRESSION is true assign TRUE\_PART to the *variable*, otherwise (when EXPRESSION is false) assign FALSE\_PART.

### EXAMPLE

*Incorrect code*

```
IIf( Close > 10, result = 7, result = 9 ); // WRONG
```

Correct code:

```
result = IIf( Close > 10, 7, 9 ); // CORRECT
```

---

### ***IIf is for arrays, WriteIf is for strings***

IIf functions should be used to handle arrays, if you need conditional text function use WriteIf instead.

### **EXAMPLE**

Incorrect code:

```
variable = IIf(Condition, "Text 1", "Text 2" ); // WRONG
```

IIf( ) function returns array, NOT STRING, so it is impossible to assign text to variable with use of IIF. Use WriteIf( ) function instead:

Correct code:

```
variable = WriteIf( condition, "Text 1", "Text 2" ); // CORRECT
```

Please note however that WriteIf function returns just single STRING, not arrays of strings, so only the **selected value** is used for evaluation.

---

### ***if-else statement needs boolean (or single numeric expression), not array***

The **if** keyword executes *statement1* if *expression* is true (nonzero); if **else** is present and *expression* is false (zero), it executes *statement2*. After executing *statement1* or *statement2*, control passes to the next statement. *Expression* must be boolean ( True/False) type (so it CANNOT be ARRAY because there would be no way to decide whether to execute *statement1* or not, if for example array was: [True,True,False,.....,False,True] )

```
if( expression )
    statement1
else
    statement2
```

### **EXAMPLE**

```
if( Close > Open ) // WRONG
    Color = colorGreen; //statement 1
else
    Color = colorRed; //statement 2

Plot(Close, "Colored Price", Color, styleCandle);
```

The above example is wrong, as both **Open** and **Close** are arrays and such expression as **Close > Open** is also an ARRAY. The solution depends on the statement. It is either possible to implement it on bar-by-bar basis, with use of FOR loop:

```
for( i = 0; i < BarCount; i++ )
```



```

{
    if( Close[ i ] > Open[ i ] ) // CORRECT
        Color[ i ] = colorGreen;
    else
        Color[ i ] = colorRed;
}

Plot( Close, "Colored Price", Color, styleCandle );

```

It is also possible in this case to use `IIf()` function:

```

Color = IIf( Close > Open, colorGreen, colorRed ); // ALSO CORRECT - working
directly on arrays
Plot( Close, "Colored Price", Color, styleCandle );

```

### Barcount vs BarIndex()

There is a fundamental difference between *BarCount* and *BarIndex()*. *BarCount* is a **numeric variable** that holds just one number (the count of elements in array). On the other hand *BarIndex()* is a function that returns ARRAY representing consecutive index of each bar.

### EXAMPLE

*Incorrect code:*

```

for ( i = 0; i < BarIndex(); i++ ) // WRONG
{
    // your formula
}

```

It's not allowed to use ARRAY inside **for** loop, and *BarIndex()* returns ARRAY. That is why it's necessary to change the formula.

*Correct code:*

```

for ( i = 0 ; i < BarCount ; i++ ) // CORRECT
{
    //your formula
}

```

### TimeFrameExpand() is required to match data with original time frame

The **TimeFrameSet()** replaces current price/volume arrays: open, high, low, close, volume, openint, avg with time-compressed bars of specified interval once you switched to a different time frame all calculations and built-in indicators operate on selected time frame. To get back to original interval call **TimeFrameRestore()** function. The **TimeFrameExpand()** is used to decompress array variables that were created in different time frame. Decompressing is required to properly display and use the array created in different time frame.

### EXAMPLE

*Incorrect code:*

```
TimeFrameSet( inWeekly );
MA14_Weekly = MA( Close, 14 );
TimeFrameRestore();
Buy = Cross( Close, MA14_Weekly ); // WRONG - Close and MA15_Weekly use different
time scales
```

The above formula is wrong, as MA14\_Weekly variable should be EXPANDED to match original timeframe. The right contents should be:

*Correct code:*

```
TimeFrameSet( inWeekly );
MA14_Weekly = MA( Close, 14 );
TimeFrameRestore();
Buy = Cross( Close, TimeFrameExpand( MA14_Weekly, inWeekly ) ); // CORRECT,
expanded weekly MA can be matched against daily close
```

## EXAMPLE 2:

*Incorrect code:*

```
TimeFrameSet( inWeekly );
MA14_Weekly = MA( Close, 14 );
TimeFrameRestore();
Buy = Cross( Close, TimeFrameExpand( MA14_Weekly, inDaily ) ); // WRONG
```

It's always necessary to indicate in TimeFrameExpand() function, which timeframe was variable calculated in. So if MA14\_Weekly was calculated in out of weekly data, **inWeekly** should be the correct parameter of TimeFrameExpand() function.

*Correct code:*

```
TimeFrameSet( inWeekly );
MA14_Weekly = MA( Close, 14 );
TimeFrameRestore();
Buy = Cross( Close, TimeFrameExpand( MA14_Weekly, inWeekly ) ); // CORRECT
```

## Portfolio Backtester Interface Reference Guide

(Updated January 22nd, 2005 to cover enhancements and additions introduced in **AmiBroker 4.68.0 BETA**)

### Basics

AmiBroker version 4.67.0 exposes new object-oriented interface to portfolio backtester allowing to control 2nd phase of the backtest. This allows multitude of applications including, but not limited to:

- position sizing based on portfolio-level equity
- implementing advanced rotational systems (you have now access to ranking arrays and can decide what trades to take after knowing which symbols scores best on bar-by-bar basis)

- adding your custom metrics to backtest and optimization statistics
- implementing custom formulas for slippage control
- advanced scaling-in/-out based on portfolio equity and other run-time stats
- advanced trading systems that use portfolio-level statistics evaluated on bar-by-bar basis to decide which trades to take

This document describes all objects, methods and properties exposed by portfolio interface.

## Requirements

To use new interface the user needs AmiBroker 4.67.0 or higher and needs to have AFL coding skills including understanding the terms: an object, method and property.

## Various approaches for various applications

The portfolio backtester interface supports various approaches to customization of backtest process that suit different applications.

- **high-level** approach (*the easiest*)
  - using Backtest() method and it runs default backtest procedure (as in old versions) – allows simple implementation of custom metrics
- **mid-level** approach
  - using PreProcess()/ProcessTradeSignal()/PostProcess() methods – allows to modify signals, query open positions (good for advanced position sizing)
- **low-level** approach (*the most complex*)
  - using PreProcess()/EnterTrade()/ExitTrade()/ScaleTrade()/UpdateStats()/HandleStops()/PostProcess() methods – provides full control over entire backtest process for hard-code programmers only

## Getting access to the interface

To access new portfolio backtester interface you need to:

- enable custom backtesting procedure by calling:

```
SetOption( "UseCustomBacktestProc", True );
```

or calling

```
SetCustomBacktestProc( "C:\\MyPath\\MyCustomBacktest.afl" );
```

in your formula

or by enabling it in Automatic Analysis->Settings window, "Portfolio" tab and specifying external custom procedure file.

- get access to backtester object by calling GetBacktesterObject() method. Note that GetBacktester method should only be called when Status("action") returns actionPortfolio:

```
if( Status("action")== actionPortfolio )
{
    // retrieve the interface to portfolio backtester
```

```

    bo = GetBacktesterObject();

    ...here is your custom backtest formula.
}

```

When using external custom procedure file you don't need to check for actionPortfolio, because external backtest procedures are called exclusively in actionPortfolio mode.

## Typing Conventions

- *bool* – italic represents the type of parameter/variable/return value (*Trade*, *Signal*, *Stats* – represent the type of object returned)
- **AddSymbols** – bold represents function / method / property name
- SymbolList – underline type face represents formal parameter
- [optional] – denotes optional parameter (that does not need to be supplied)
- *variant* – represent variable type that can be either string or a number

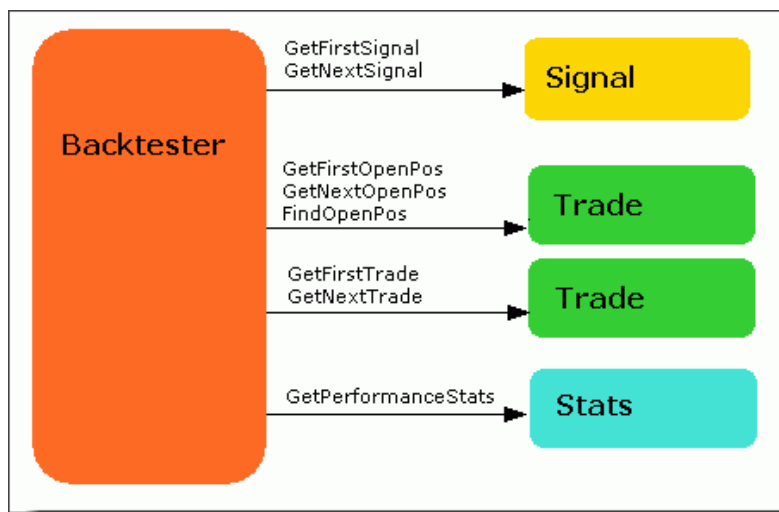
Despite the fact that interface handles integer data type such as long, short, bool and two different floating point types: float and double, the AFL itself converts all those data types to float because AFL treats all numbers as floats (32-bit IEEE floating point numbers).

## Objects

The interface exposes the following objects:

- Backtester object
- Signal object
- Trade object
- Stats object

The only object directly accessible from AFL is Backtester object, all other objects are accessible by calling Backtester object methods as shown in the picture below.



**Backtester object**

Backtester object allows to control backtest process (process signals, enter/exit/scale trades) and get access to signal list, open position and trade list and to performance statistics object.

Methods:

- **bool AddCustomMetric**( *string Title*, *variant Value*, [optional] *variant LongOnlyValue*, [optional] *variant ShortOnlyValue*, [optional] *variant DecPlaces* = 2 )

This method adds custom metric to the backtest report, backtest "summary" and optimization result list. Title is a name of the metric to be displayed in the report, Value is the value of the metric, optional arguments LongOnlyValue, ShortOnlyValue allow to provide values for additional long/short-only columns in the backtest report. Last argument DecPlaces controls how many decimal places should be used to display the value.

- **bool Backtest**( [optional] **bool NoTradeList** )

This high-level method performs default portfolio backtest procedure in single call. It should be used if you just need to obtain custom metrics and do not want to change the way backtest is performed. (Version 4.68.0 and above): If optional parameter NoTradeList is set to True, then trade list is not generated automatically. This is useful if you want to add some per-trade metrics. Once you add them, you can generate trade list with your metrics using **ListTrades** method.

- **long EnterTrade**( *long Bar*, *string Symbol*, *bool bLong*, *float Price*, *float PosSize*, [optional] *variant PosScore*, [optional] *variant RoundLotSize*, [optional] *variant MarginDeposit*, [optional] *variant TickSize*, [optional] *variant PointValue* )

Low-level method that enters trade on any symbol. This allows to take trades even on symbols that have no corresponding signals. If values for optional parameters are not provided then AmiBroker uses values defined in Symbol->Information window

- **long ExitTrade**( *long Bar*, *string Symbol*, *float Price*, [optional] *variant ExitType* )

Low-level method that exits trade on any symbol. This method searches open trade list and if there is no open trade on given symbol it does nothing. Optional ExitType parameter specifies the reason for exit (0 – regular exit, 1 – max. loss, 2 – profit, 3 – trail, 4 – N-bar, 5 – ruin)

- **Trade FindOpenPos**( *string Symbol* )

This method looks for the Symbol in open position list and returns matching trade object if it finds one or returns null object otherwise.

- **Trade GetFirstOpenPos**()

This method returns first *Trade* object from open position list

- **Signal GetFirstSignal**( *long Bar* )

This method returns first trading *Signal* object for given Bar

- Trade **GetFirstTrade()**

This method returns first *Trade* object from closed trade list

- Trade **GetNextOpenPos()**

This method returns next *Trade* object from open positions list. You should call `GetFirstOpenPos` before calling this method for the first time. Returns null object when no more open positions are found.

- Signal **GetNextSignal( long Bar )**

This method returns next *Signal* object from closed signal list of given Bar. You should call `GetFirstSignal` before calling this method for the first time. Returns null object when no more signals are found.

- Trade **GetNextTrade()**

This method returns next *Trade* object from closed trade list. You should call `GetFirstTrade` before calling this method for the first time. Returns null object when no more trades are found.

- Stats **GetPerformanceStats( long Type )**

Calculates built-in statistics and metrics and returns *Stats* object. Type parameter specifies what trades should be counted in. Type = 0 means all trades, Type = 1 means long-only, Type = 2 means short-only.

- **HandleStops( long Bar )**

This low-level method handles automatic stops (applystops). ***This method MUST NOT be used in high-level and mid-level approaches.*** In low-level mode you should call this method once for each bar inside trading loop.

- **ListTrades()**

(Version 4.68.0 and above) This outputs trades to the result list of Automatic Analysis window. Usually this function does NOT need to be called because ***Backtest()*** method by default lists trades already. This function should only be used when you disabled trade listing in `Backtest()` method to add some custom per-trade metrics.

- **PostProcess()**

This mid-level and low-level method performs final processing required to complete backtest correctly. Among other things it frees price cache, closes out any open trades and outputs trade list to the Automatic Analysis window. It should NOT be used when you call `Backtest()` method because `Backtest()` already performs necessary processing.

- **PreProcess()**

This mid-level and low-level method performs initial processing required to perform backtest correctly. Among other things it initializes price cache and sets up initial variables. It should NOT be used when you call `Backtest()` method because `Backtest()` already performs necessary processing.

- **bool *ProcessTradeSignals***( long Bar )

This mid-level method processes all trading signals for given bar. It should be called once per every bar in your custom backtesting loop.

- **RawTextOutput**( string Text )

(Version 4.68.0 and above) This method outputs any string to the Automatic Analysis result list at the time of the call. The user can output text formatted in multiple columns using \t (tab) character.

- **long *ScaleTrade***(long Bar, string Symbol, bool blncrease, float Price, float PosSize, [optional] variant Deposit)

Low-level method that scales trade on any symbol. This method searches open trade list and if there is no open trade on given symbol it does nothing. Optional Deposit parameter specifies margin deposit for futures, if not given then Price parameter is used.

- **UpdateStats**( long Bar, long TimeInsideBar )

Low-level method that updates equity, exposure, trade excursions (for MAE/MFE calculations) and other internal variables required for correct calculation of statistics. **You must NOT use this function in high-level and mid-level approaches.** TimeInsideBar parameter specifies intraday time position. TimeInsideBar = 0 means opening of the bar, TimeInsideBar = 1 means middle of the bar, TimeInsideBar = 2 means end of bar. As certain internal calculations depend on end-of-bar calculations, this method must be called once and only once with TimeInsideBar parameter set to 2 at the end of processing of every bar inside in your custom backtesting loop. May be called zero or more times for every bar inside backtesting loop with TimeInsideBar parameter set to 0 (zero) or 1.

Properties:

- **double *Cash***

available funds (cash) in your portfolio

- **double *Equity***

current portfolio-vele Equity (read-only property)

- **double *InitialEquity***

funds that are available at the beginning of the backtest

- **double *MarginLoan***

loan amount (only if you are using margin account) (read-only property)

**Signal object**

Signal object represents trading signal (buy/sell/short/cover) or ranking array element generated by AmiBroker during first phase of backtest when your formula is executed on every symbol under test. During this first

phase scan AmiBroker collects data from buy/sell/short/cover signal, price, position size and score arrays, performs sorting of signals and put top-ranked entry signals and all scale and exit signals into the list. Separate list of trading signals is maintained for every bar. Signal list is sorted so first entry signals appear (top ranked first) and after that scaling and exit signals follow. To conserve memory AmiBroker stores only (2\*MaxOpenPositons) top-ranked entry signals per bar. It keeps however all exit and scaling signals. Once first phase is completed and backtester enters 2nd phase (real backtest) it iterates through bars and through all signals within given bar and executes trades based on this signals.

To iterate through signal list you should use GetFirstSignal() / GetNextSignal() methods of Backtester object, as shown below:

```
// retrieve the interface to portfolio backtester
bo = GetBacktesterObject();

for( sig = bo.GetFirstSignal(); sig; sig = bo.GetNextSignal() )
{
    if( sig.IsEntry() )
    {
        // handle entry signal
        ....
    }
}
```

Methods:

- **bool IsEntry()**

True if this is entry signal, False otherwise

- **bool IsExit()**

True if this is exit signal, False otherwise

- **bool IsLong()**

True if this is long entry (buy) or long exit (sell) or scale-in signal, False otherwise

- **bool IsScale()**

True if this is scale-in or scale-out signal, False otherwise

Properties:

- **float MarginDeposit**

margin deposit (for futures)

- **float PointValue**

point value (for futures, currencies)



- *float* **PosScore**

position score

- *float* **PosSize**

requested position size (positive numbers mean dollar value, negative values mean percent of portfolio equity)

- *float* **Price**

entry/exit/scale price

- *short int* **Reason**

this specifies reason of exit ( 0 – regular exit, 1 – max. loss, 2 – profit, 3 – trail, 4 – N-bar, 5 – ruin )

- *float* **RoundLotSize**

round lot size

- *string* **Symbol**

symbol of security

- *float* **TickSize**

tick size (minimum price change)

- *short int* **Type**

this specifies signal type ( 0 – rank (rotational systems only), 1 – buy, 2 – sell, 3 – short, 4 – cover, 5 – scale-in, 6 – scale-out )

## Trade object

Trade object represents either currently open position (open trade) or closed trade. AmiBroker maintains 2 lists of trades: open position list (accessible using GetFirstOpenPos/GetNextOpenPos methods of backtester object) and closed trade lists (accessible using GetFirstTrade/GetNextTrade methods of the backtester objects). Once open position is closed by the backtester it is automatically moved from open position list to trade list. When backtest is completed (after PostProcess call) AmiBroker closes out all open positions, so trade list includes all trades. You can access both lists any time during backtest, you can also access trade list after completion to generate trade-related stats.

To iterate through open position list you should use GetFirstOpenPos() / GetNextOpenPos() methods of Backtester object, as shown below:

```
// 'bo' variable holds Backtester object retrieved earlier

for( openpos = bo.GetFirstOpenPos(); openpos; openpos = bo.GetNextOpenPos() )
```

```
{
    // openpos variable now holds Trade object
}
```

To iterate through closed trade list you should use `GetFirstTrade()` / `GetNextTrade()` methods of `Backtester` object, as shown below:

```
for( trade = bo.GetFirstTrade(); trade; trade = bo.GetNextTrade() )
{
    // trade variable now holds Trade object
}
```

Methods:

- **long *AddCustomMetric*( string Title, variant Value )**

(Version 4.68.0 BETA and above) This method adds PER-TRADE custom metric to the trade list only. Title is a name of the metric to be displayed in the report, Value is the value of the metric. When using this function you have to ensure that you the same metrics in the same order to every trade. Otherwise output may be messed up. Note that in contrast to ***Backtester.AddCustomMetric*** method that is usually called after `PostProcess`, the ***Trade.AddCustomMetric*** should be called before ***PostProcess*** call because ***PostProcess*** lists trades. Using ***Trade.AddCustomMetric*** after ***PostProcess*** gives no result, because trades are already listed. Also if you are using ***Backtester.Backtest()*** method you should call it with `NoTradeList` parameter set to `True`, add your per-trade metrics and then call ***ListTrades*** to allow your custom metrics to be included in the output.

- **float *GetCommission*( [optional] bool InclExit )**

retrieves commission paid for that trade (includes all scale in/out commissions). Depending on `InclExit` parameter the function returns commission including (`True`, default) or excluding (`False`) exit commission.

- **double *GetEntryValue*()**

retrieves dollar entry value of the trade

- **float *GetMAE*()**

retrieves trade's Maximum Adverse Excursion in percent

- **float *GetMFE*()**

retrieves trade's Maximum Favorable Excursion in percent

- **double *GetPercentProfit*()**

retrieves current percent profit of the trade

- **double *GetPositionValue*( )**

retrieves current dollar value of the position.

- **float *GetPrice*( long *Bar*, string *Field* )**

(Version 4.68.0 BETA and above) provides quick access to price arrays of open positions. *Bar* parameter represents the data bar to query price for, *Field* parameter specifies which price field you want to get, allowable values are:

"O" (Open)  
 "H" (High)  
 "L" (Low)  
 "C" (Close)  
 "F" (Fx currency rate)

NOTES:

1. GetPrice method is available for OPEN POSITIONS only, when called on closed trade returns Null value
2. Open Interest field is NOT available via GetPrice
3. Bar must be between 0..BarCount-1, otherwise exception will occur

- **double *GetProfit*()**

retrieves current dollar (point) profit of the trade

Properties:

- **long *BarsInTrade***

bars spent in trade (counting starts from 0)

Note however that the value of zero is available only when trade is just opened in "low-level" approach, so normally you would see numbers  $\geq 1$  (all other reporting in AB remains as it was, so enter today and exit tomorrow counts as 2-bar trade)

- **float *EntryDateTime***

entry date/time in internal AmiBroker format (the same as used by AFL function *DateTime()*)

- **float *EntryFxRate***

entry foreign exchange currency rate, if any scaling-in occurred this holds average entry fx rate

- **float *EntryPrice***

entry price, if any scaling-in occurred this holds average entry price

- **float *ExitDateTime***

exit date/time in internal AmiBroker format (the same as used by AFL function *DateTime()*)

- *float* **ExitFxRate**

exit foreign exchange currency rate, if any scaling-out occurred this holds average exit fx rate

- *float* **ExitPrice**

exit price, if any scaling-out occurred this holds average exit price

- *double* **Handle**

internal handle value that allows to uniquely identify and manage (for example exit or scale in/out) multiple trades open on the same symbol at the same time. It can be passed to ExitTrade / ScaleTrade instead of the symbol.

- *bool* **IsLong**

True if trade is long, False otherwise

- *bool* **IsOpen**

True if trade is open, False otherwise

- *float* **MarginDeposit**

initial margin deposit

- *double* **MarginLoan**

loan amount used for this trade

- *float* **PointValue**

point value (for futures / currencies)

- *float* **RoundLotSize**

round lot size

- *float* **Score**

entry score

- *float* **Shares**

number of shares / contracts

- *string* **Symbol**

symbol of the security

- *float* **TickSize**

tick size (minimum price change)

**Stats object**

Stats object provides the access to built-in backtester statistics and metrics. Metrics are usually calculated once backtest is completed but it is also possible to calculate metrics during backtest. To calculate current metrics and get the access to them simply call `GetPerformanceStats` method of `Backtester` object. Please note that if you calculate statistics in the middle of the backtest they will include only closed trades.

To calculate and access stats use the following code:

```
// 'bo' variable holds Backtester object retrieved earlier

stats = bo.GetPerformanceStats( 0 );
```

Methods:

- **double *GetValue*( string MetricName )**

retrieves the value of a metric, MetricName can be one of the following:

```
"InitialCapital" ,
"EndingCapital"
"NetProfit"
"NetProfitPercent"
"ExposurePercent"
"NetRAR"
"CAR"
"RAR"

"AllQty"
"AllPercent"
"AllAvgProfitLoss"
"AllAvgProfitLossPercent"
"AllAvgBarsHeld"

"WinnersQty"
"WinnersPercent"
"WinnersTotalProfit"
"WinnersAvgProfit"
"WinnersAvgProfitPercent"
"WinnersAvgBarsHeld"
"WinnersMaxConsecutive"
"WinnersLargestWin"
"WinnersLargestWinBars"

"LosersQty"
"LosersPercent"
"LosersTotalLoss"
"LosersAvgLoss"
"LosersAvgLossPercent"
"LosersAvgBarsHeld" ,
```

"LosersMaxConsecutive"  
 "LosersLargestLoss"  
 "LosersLargestLossBars"  
  
 "MaxTradeDrawdown"  
 "MaxTradeDrawdownPercent"  
 "MaxSystemDrawdown"  
 "MaxSystemDrawdownPercent"  
 "RecoveryFactor"  
 "CAR/MDD"  
 "RAR/MDD"  
 "ProfitFactor"  
 "PayoffRatio"  
 "StandardError"  
 "RRR"  
 "UlcerIndex"  
 "UlcerPerformanceIndex"  
 "SharpeRatio"  
 "KRatio"

Properties:

–none–

### Further information

Examples and more documentation can be found in [this Houston presentation covering custom backtester interface \(300 KB PDF format\)](#) and the Knowledge Base: <http://www.amibroker.com/kb/category/afl/custom-backtest/>

## How to add user-defined metrics to backtest/optimization report

One of the new additions in 4.67.x/4.68.x BETA is [portfolio backtester programming interface](#) providing full control of 2nd phase of portfolio backtest. This allows multitude of applications including, but not limited to:

- user-defined metrics (appear as an additional column in "summary" backtest result list, in optimization result and as a new row in backtest report "statistics" page, as well as per-trade metrics)
- access to portfolio-level equity when backtest loop is run – allows for example complex position sizing based on portfolio equity
- read/write access to portfolio cash – allows adding funds to portfolio
- read/write access on bar-by-bar basis to trading signals generated in 1st backtest phase allows reading ranking array and modifying signal price (for example custom slippage formulas), position size, etc
- access to list of currently open positions, each position can be queried for various properties including profit, MAE/MFE, bars in trade, etc
- access to list of closed trades, each closed trade can be queried for various properties including profit, MAE/MFE, bars in trade, etc
- three different levels of programming:
  - ◆ high-level – using Backtest() method and it runs default backtest procedure (as in old versions) – great for adding custom metrics

- ◆ mid-level – using PreProcess()/ProcessTradeSignal()/PostProcess() methods – allows to modify signals, query open positions (good for advanced position sizing)
- ◆ low-level – using PreProcess()/EnterTrade()/ExitTrade()/ScaleTrade()/UpdateStats()/HandleStops()/PostProcess() methods provides full control over backtest process for hard-code programmers

Technical reference of new interface is available [here](#), in this chapter we will just focus on some practical examples.

### **Adding user-defined metrics**

#### *Example 1*

Let's start with the easiest application: in the very first example I will show you how to add user-defined metric to portfolio report and optimization result list.

In the first step we will add Expectancy to backtest and optimization report. There is some discussion about how expectancy should be calculated but the easiest formula for it is:

Expectancy (\$) = %Winners \* AvgProfit – %Losers \* AvgLoss

or (the other way of calculating the same)

Expectancy (\$) = (TotalProfit – TotalLoss) / NumberOfTrades = NetProfit / NumberOfTrades

Let us start with this simple formulation. With this approach expectancy simply tells us expected profit per trade in dollars. The custom backtest formula that implements this user-defined metric looks as follows:

```
/* First we need to enable custom backtest procedure and
** tell AmiBroker to use current formula
*/

SetCustomBacktestProc( "" );

/* Now custom-backtest procedure follows */

if( Status("action") == actionPortfolio )
{
    bo = GetBacktesterObject();

    bo.Backtest(); // run default backtest procedure

    st = bo.GetPerformanceStats(0); // get stats for all trades

    // Expectancy calculation (the easy way)
    // %Win * AvgProfit - %Los * AvgLos
    // note that because AvgLos is already negative
    // in AmiBroker so we are adding values instead of subtracting them
    // we could also use simpler formula NetProfit/NumberOfTrades
    // but for the purpose of illustration we are using more complex one :- )
    expectancy =
    st.GetValue( "WinnersAvgProfit" ) * st.GetValue( "WinnersPercent" ) / 100 +
```

```

st.GetValue( "LosersAvgLoss" ) * st.GetValue( "LosersPercent" ) / 100;

// Here we add custom metric to backtest report
bo.AddCustomMetric( "Expectancy ($)", expectancy );
}

// your trading system here
fast = Optimize( "fast", 12, 5, 20, 1 );
slow = Optimize( "slow", 26, 10, 25, 1 );
Buy=Cross(MACD(fast,slow),Signal(fast,slow));
Sell=Cross(Signal(fast,slow),MACD(fast,slow));

```

First we need to tell AmiBroker to use custom backtest formula instead of built-in one. We are doing so by calling SetCustomBacktestProc. First parameter defines the path to the custom backtest formula (which can be stored in some external file, independent from actual trading system). If we provide empty string there, we are telling AmiBroker to use current formula (the same which is used for trading system).

In the next line we have "if" statement that enters custom backtest formula if the analysis engine is in actionPortfolio (2nd phase of portfolio backtest) stage. This is important as formula is executed in both scanning phase (when trading signals are generated) and in actual portfolio backtest phase. "if" statement allows us to enter custom backtest procedure part only when analysis engine is in actual backtesting phase.

In the next line we obtain the access to backtester programming interface by calling GetBacktesterObject function. This returns Backtester object that is used to access all functionality of new interface (more details on objects available see: <http://www.amibroker.com/docs/ab401.html>)

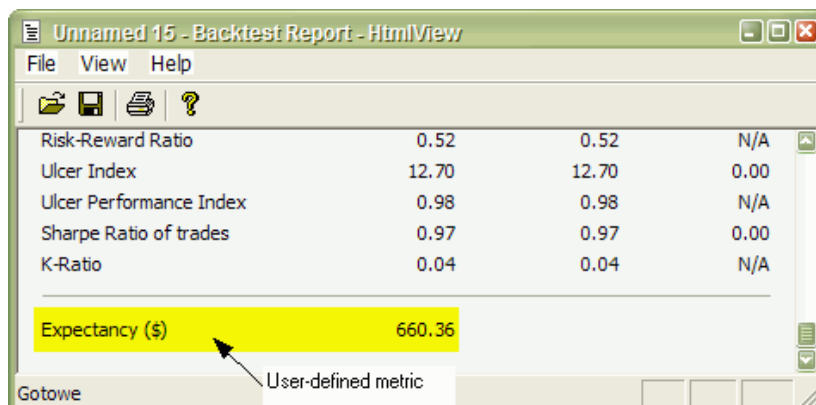
Later we obtain access to built-in metrics by calling GetPerformanceStats method of backtester object. This method returns Statistics object that allows us to access any built-in metric by calling GetValue method.

As a next step we calculate expectancy value from built-in metrics retrieved using GetValue method. For the list of metrics supported by GetValue method please check: <http://www.amibroker.com/docs/ab401.html>

In the final step we simply add our custom metric to the report by calling AddCustomMetric function of Backtester object. The first parameter is the name of the metric, the second is the value.

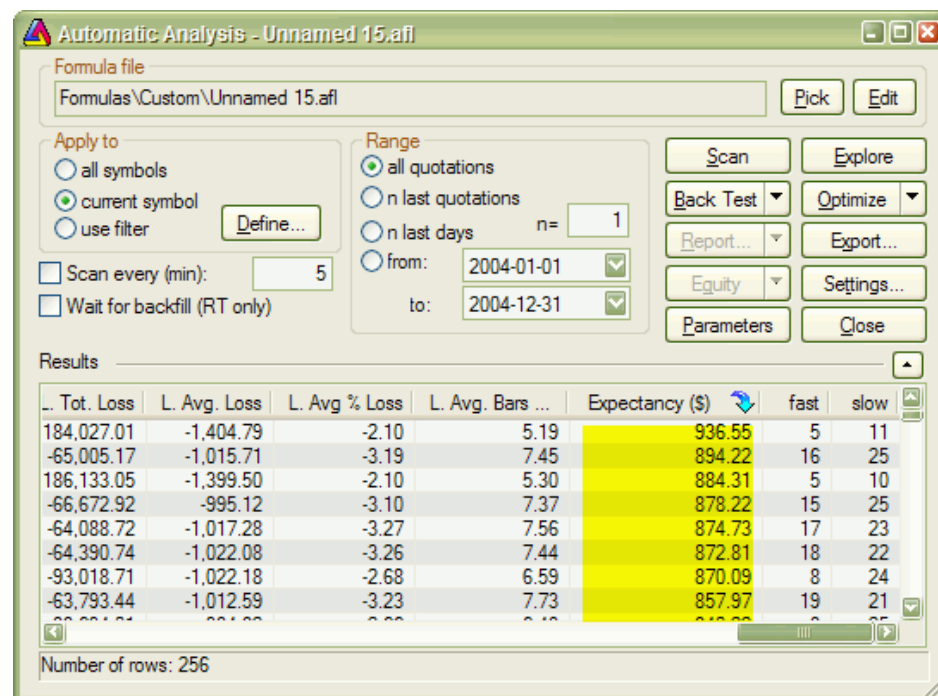
After "if"-statement implementing our custom backtest procedure usual trading system rules follow.

Now when you run Backtest and click Report button in Automatic Analysis window you will see your custom metric added at the bottom of statistics page:

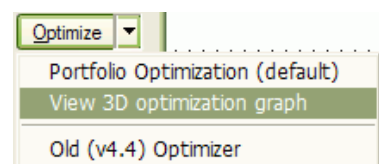


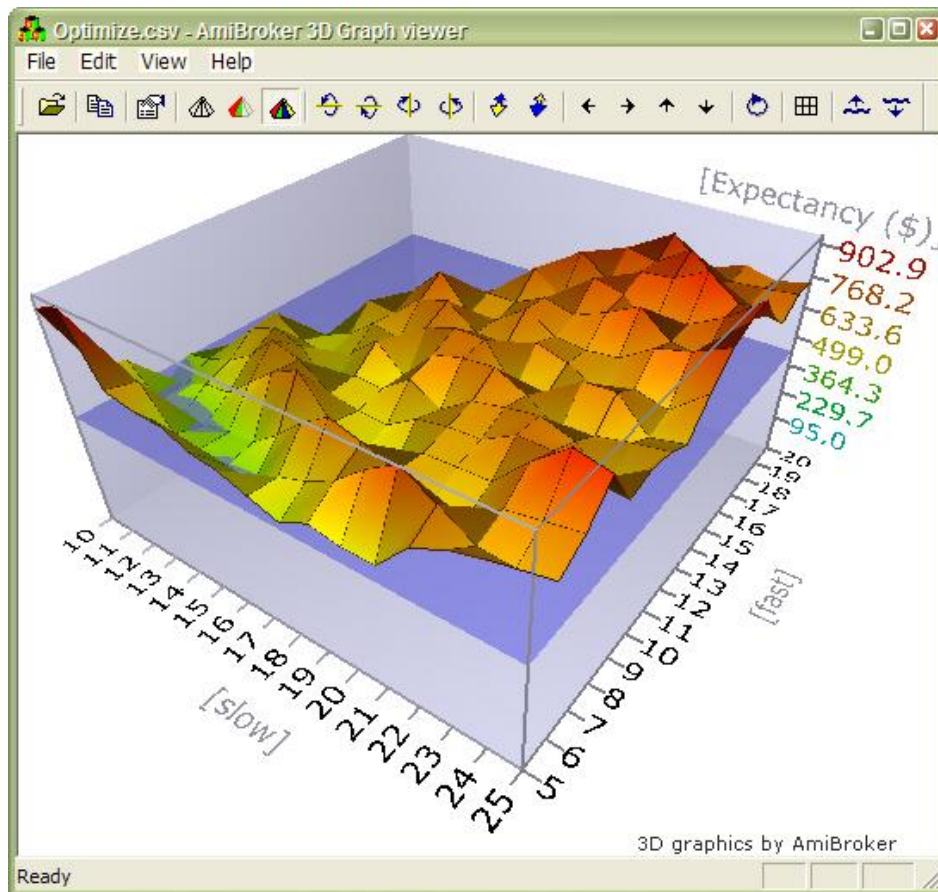


User-defined metric also appears in the Optimization result list:



When you click on the custom metric column, the optimization results will be sorted by your own metric and you will be able to display 3D chart of your user-defined metric plotted against optimization variables.





### Example 2

Some people point out that this simple method of calculating expectancy works well only with constant position size. Otherwise, with variable position sizing and/or compounding, larger trades weight more than smaller trades and this leads to misleading expectancy values. To address this problem one could calculate expectancy for example as expected profit per \$100 invested. To do calculate such statistic, one needs to iterate through trades, summing up profits per \$100 unit, and dividing this sum by the number of trades. Appropriate formula follows:

```
/* First we need to enable custom backtest procedure and
** tell AmiBroker to use current formula
*/

SetCustomBacktestProc( "" );

/* Now custom-backtest procedure follows */

if( Status("action") == actionPortfolio )
{
    bo = GetBacktesterObject();

    bo.Backtest(); // run default backtest procedure
}
```

```

SumProfitPer100Inv = 0;
NumTrades = 0;

// iterate through closed trades first
for( trade = bo.GetFirstTrade(); trade; trade = bo.GetNextTrade() )
{
    // here we sum up profit per $100 invested
    SumProfitPer100Inv = SumProfitPer100Inv + trade.GetPercentProfit();
    NumTrades++;
}

// iterate through eventually still open positions
for( trade = bo.GetFirstOpenPos(); trade; trade = bo.GetNextOpenPos() )
{
    SumProfitPer100Inv = SumProfitPer100Inv + trade.GetPercentProfit();
    NumTrades++;
}

expectancy2 = SumProfitPer100Inv / NumTrades;

bo.AddCustomMetric( "Expectancy (per $100 inv.)", expectancy2 );
}

// your trading system here
fast = Optimize( "fast", 12, 5, 20, 1 );
slow = Optimize( "slow", 26, 10, 25, 1 );
Buy=Cross(MACD(fast,slow),Signal(fast,slow));
Sell=Cross(Signal(fast,slow),MACD(fast,slow));

```

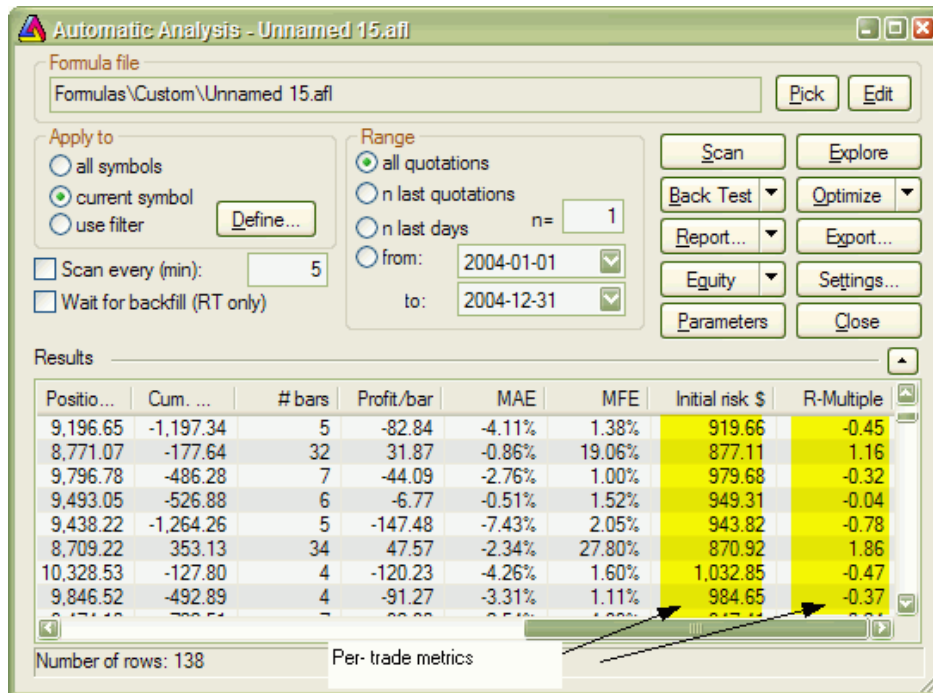
The only difference between this and previous formula is that we do not use built-in metrics to calculate our own expectancy figure. Instead we sum up all percentage profits of each trade (which are equivalent to dollar profits from \$100 unit investment) and at the end divide the sum by the number of trades. Summing up is done inside the "for" loop. GetFirstTrade/GetNextTrade function pair of the backtester object allows us to step through the list of closed trades. We use two loops (second loop uses GetFirstOpenPos/GetNexOpenPos) because there may be some open positions left at the end of the backtest. If we wanted to include only closed trades then we could remove second "for" loop.

After running this code we find out that expectancy calculated this way even adjusted to initial equity (by multiplying by factor InitialEquity/\$100) is smaller than expectancy calculated in the first example. This shows that "easy" method of expectancy calculation (from example 1) may lead to overly optimistic results.

### Example 3

Some Van Tharp followers prefer yet slightly differnt "twist" of expectancy measure. They express expectancy in terms of expected profit per "unit of risk". The profit is then expressed in terms of R-multiples, where 1R is defined as the amount risked per trade. The amount risked is the maximum amount of money you can lose, and most often it is set by the amount of maximum loss stop (or trailing stop). According to Tharp, the easiest way to calculate expectancy is simply to add up all your R-multiples and net them out by subtracting the negative R-multiples from the positive ones, then divide by the no. of trades. This gives you your expectancy per trade.

This is very similar to approach presented in example 2, but for the calculations we do not use the value of the trade but rather risk per trade. The risk depends on the stop we use in our trading system. For simplicity in this example we have used 10% max. loss stop. In this example we also add per-trade metrics for better illustration of how R-multiples are calculated. Per-trade metrics appear in each row of the trade list in the backtest results.



The formula that implements this kind of expectancy measure follows:

```

/* First we need to enable custom backtest procedure and
** tell AmiBroker to use current formula
*/

SetCustomBacktestProc( "" );

MaxLossPercentStop = 10; // 10% max. loss stop

/* Now custom-backtest procedure follows */
if( Status("action") == actionPortfolio )
{
    bo = GetBacktesterObject();

    bo.Backtest(1); // run default backtest procedure

    SumProfitPerRisk = 0;
    NumTrades = 0;

    // iterate through closed trades first
    for( trade = bo.GetFirstTrade(); trade; trade = bo.GetNextTrade() )
    {
        // risk is calculated as the maximum value we can loose per trade
        // in this example we are using max. loss stop
    }
}

```

```

// it means we can not lose more than (MaxLoss%) of invested amount
// hence ris

Risk = ( MaxLossPercentStop / 100 ) * trade.GetEntryValue();
RMultiple = trade.GetProfit()/Risk;

trade.AddCustomMetric("Initial risk $", Risk );
trade.AddCustomMetric("R-Multiple", RMultiple );

SumProfitPerRisk = SumProfitPerRisk + RMultiple;
NumTrades++;
}

expectancy3 = SumProfitPerRisk / NumTrades;

bo.AddCustomMetric( "Expectancy (per risk)", expectancy3 );

bo.ListTrades();

}

// your trading system here

ApplyStop( stopTypeLoss, stopModePercent, MaxLossPercentStop );

fast = Optimize("fast", 12, 5, 20, 1 );
slow = Optimize("slow", 26, 10, 25, 1 );
Buy=Cross(MACD(fast,slow),Signal(fast,slow));
Sell=Cross(Signal(fast,slow),MACD(fast,slow));

```

The code is basically very similar to example 2. There are only few differences. First is that we call Backtest method with NoTradeList parameter set to 1. This way we disable default trade listing, so we can add custom per-trade metrics and list trades later by calling ListTrades method. Later we iterate through trades and calculate risk based on trade entry value and amount of max. loss stop used. The RMultiple is then calculated as trade profit divided by the amount risked per trade. Both risk and r-multiple are then added as custom per-trade metrics (note that we are calling AddCustomMetric method of **Trade** object here). Later on we do remaining calculations. At the end of the custom backtest procedure we are adding custom backtest metric (this time calling AddCustomMetric method of **Backtester** object), and after that we trigger listing of the trades using ListTrades method. For simplicity we ignore any open positions that may have left at the end of analysis period. The only change to the trading system itself was addition of maximum loss stop (ApplyStop line).

## Conclusion

A new portfolio backtester programming interface provides ability to add user-defined statistics of any kind, allowing the user to move the analysis of backtesting results to completely new level.

## Using low-level graphics functions

Completely new low-level graphic AFL interface allows complete flexibility in creating any kind of user-defined display.

The interface mimics closely Windows GDI API, with same names for most functions for easier use for GDI-experienced programmers. The only differences are:

1. compared to Windows GDI all functions are prefixed with 'Gfx'
2. pen/brush/font creation/selection is simplified to make it easier to use and you don't need to care about deletion of GDI objects
3. three overlay modes are available so you can mix low-level graphics with regular Plot() statements (mode = 0 (default) – overlay low-level graphic on top of charts, mode = 1 – overlay charts on top of low-level graphic, mode = 2 – draw only low-level graphic (no regular charts/grid/titles/etc))

All functions use PIXELS as co-ordinates (when used on screen). For printouts and metafiles pixels are mapped to logical units to match higher resolution of printers. Use Status("pxwidth") and Status("pxheight") to find pixel dimensions of drawing surface.

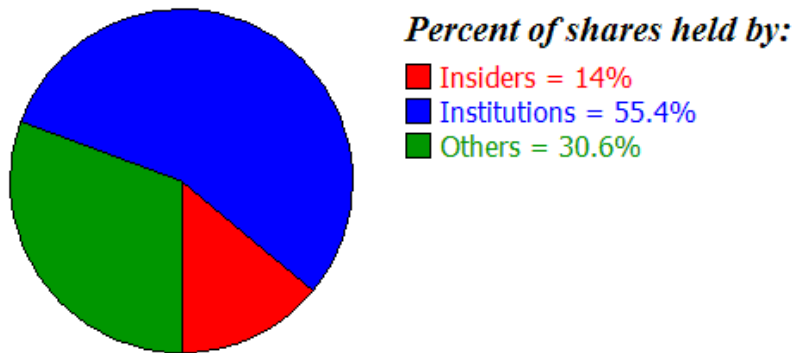
Available low-level gfx functions (click on the links for detailed explanation):

[GfxMoveTo](#)( x, y )  
[GfxLineTo](#)( x, y )  
[GfxSetPixel](#)( x, y, color )  
[GfxTextOut](#)( "text", x, y )  
[GfxSelectPen](#)( color, width = 1, penstyle = penSolid )  
[GfxSelectSolidBrush](#)( color )  
[GfxSelectFont](#)( "facename", pointsize, weight = fontNormal, italic = False, underline = False, orientation = 0 )  
[GfxRectangle](#)( x1, y1, x2, y2 )  
[GfxRoundRect](#)( x1, y1, x2, y2, x3, y3 )  
[GfxPie](#)( x1, y1, x2, y2, x3, y3, x4, y4 )  
[GfxEllipse](#)( x1, y1, x2, y2 )  
[GfxCircle](#)( x, y, radius )  
[GfxChord](#)( x1, y1, x2, y2, x3, y3, x4, y4 )  
[GfxArc](#)( x1, y1, x2, y2, x3, y3, x4, y4 )  
[GfxPolygon](#)( x1, y1, x2, y2, ... )  
[GfxPolyline](#)( x1, y1, x2, y2, ... )  
[GfxSetTextColor](#)( color )  
[GfxSetTextAlign](#)( align )  
[GfxSetBkColor](#)( color )  
[GfxSetBkMode](#)( bkmode )  
[GfxGradientRect](#)( x1, y1, x2, y2, fromcolor, tocolor )  
[GfxDrawText](#)( "text", left, top, right, bottom, format = 0 )  
[GfxSetOverlayMode](#)( mode = 0 )

### Usage examples:

Example 1. Pie-chart showing percentage holding of various kinds of shareholders

Here is how it looks:



Created with AmiBroker - advanced charting and technical analysis software. <http://www.amibroker.com>

Here is the formula:

```
// OverlayMode = 2 means that nothing except
// low-level gfx should be drawn
// there will be no grid, no title line, no plots
// and nothing except what we code using Gfx* calls
GfxSetOverlayMode(2);

HInsiders = GetFnData("InsiderHoldPercent");
HInst = GetFnData("InstitutionHoldPercent");

function DrawPiePercent( x, y, radius, startpct, endpct )
{
    PI = 3.1415926;
    sa = 2 * PI * startpct / 100;
    ea = 2 * PI * endpct / 100;
    xsa = x + radius * sin( sa );
    ysa = y + radius * cos( sa );
    xea = x + radius * sin( ea );
    yea = y + radius * cos( ea );

    GfxPie( x - radius, y - radius, x + radius, y + radius, xsa, ysa, xea, yea );
}

radius = 0.45 * Status("pxheight"); // get pixel height of the chart and use 45%
for pie chart radius
textoffset = 2.4 * radius;
GfxSelectSolidBrush( colorRed );
DrawPiePercent( 1.1*radius, 1.1*radius, radius, 0, HInsiders );
GfxRectangle( textoffset, 42, textoffset+15, 57 );
GfxSelectSolidBrush( colorBlue );
DrawPiePercent( 1.1*radius, 1.1*radius, radius, HInsiders, HInst + HInsiders );
GfxRectangle( textoffset, 62, textoffset+15, 77 );
GfxSelectSolidBrush( colorGreen );
DrawPiePercent( 1.1*radius, 1.1*radius, radius, HInst + HInsiders, 100 );
GfxRectangle( textoffset, 82, textoffset+15, 97 );
```

```

GfxSelectFont("Times New Roman", 16, 700, True );
GfxTextOut("Percent of shares held by:", textoffset , 10 );
GfxSelectFont("Tahoma", 12 );
GfxSetTextColor( colorRed );
GfxTextOut( "Insiders = " + HInsiders + "%", textoffset + 20, 40 );
GfxSetTextColor( colorBlue );
GfxTextOut( "Institutions = " + HInst + "%", textoffset + 20, 60 );
GfxSetTextColor( colorGreen );
GfxTextOut( "Others = " + ( 100 - (HInst+HInsiders) ) + "%", textoffset + 20, 80
);

GfxSelectFont("Tahoma", 8 );

```

#### Example 2. Formatted (table-like) output sample using low-level gfx functions

// formatted text output sample via low-level gfx functions

```

CellHeight = 20;
CellWidth = 100;
GfxSelectFont( "Tahoma", CellHeight/2 );

function PrintInCell( string, row, Col )
{
  GfxDrawText( string, Col * CellWidth, row * CellHeight, (Col + 1 ) * CellWidth,
  (row + 1 ) * CellHeight, 0 );
}

PrintInCell( "Open", 0, 0 );
PrintInCell( "High", 0, 1 );
PrintInCell( "Low", 0, 2 );
PrintInCell( "Close", 0, 3 );
PrintInCell( "Volume", 0, 4 );

GfxSelectPen( colorBlue );
for( i = 1; i < 10 &i < BarCount; i++ )
{
  PrintInCell( StrFormat("%g", O[ i ] ), i, 0 );
  PrintInCell( StrFormat("%g", H[ i ] ), i, 1 );
  PrintInCell( StrFormat("%g", L[ i ] ), i, 2 );
  PrintInCell( StrFormat("%g", C[ i ] ), i, 3 );
  PrintInCell( StrFormat("%g", V[ i ] ), i, 4 );
  GfxMoveTo( 0, i * CellHeight );
  GfxLineTo( 5 * CellWidth, i * CellHeight );
}
GfxMoveTo( 0, i * CellHeight );
GfxLineTo( 5 * CellWidth, i * CellHeight );

```



```

for( Col = 1; Col < 6; Col++ )
{
  GfxMoveTo( Col * CellWidth, 0);
  GfxLineTo( Col * CellWidth, 10 * CellHeight );
}

```

```

Title=" ";

```

Example 3. Low-level graphics demo featuring pie section, polygon, color-wheel, animated text and chart overlay

```

// overlay mode = 1 means that
// Low-level gfx stuff should come in background
GfxSetOverlayMode(1);

Plot(C, "Close", colorBlack, styleCandle );

PI = 3.1415926;

k = (GetPerformanceCounter()/100)%256;
for( i = 0; i < 256; i++ )
{
  x = 2 * PI * i / 256;

  GfxMoveTo( 100+k, 100 );
  GfxSelectPen( ColorHSB( ( i + k ) % 256, 255, 255 ), 4 );
  GfxLineTo( 100 +k+ 100 * sin( x ), 100 + 100 * cos( x ) );
}

GfxSelectFont( "Tahoma", 20, 700 );
GfxSetBkMode(1);
GfxSetTextColor(colorBrown);
GfxTextOut( "Testing graphic capabilities", 20, 128-k/2 );

GfxSelectPen( colorRed );
GfxSelectSolidBrush( colorBlue );
GfxChord(100,0,200,100,150,0,200,50);

//GfxPie(100,0,200,100,150,0,200,50);
GfxSelectPen( colorGreen, 2 );
GfxSelectSolidBrush( colorYellow );
GfxPolygon(250,200,200,200,250,0,200,50);

RequestTimedRefresh(1);

```

Example 4. Low-level graphic positioning – shows how to align built-in plots() with the low-level graphics. Note that if scale changes (pxheight changes) due to new data or different zoom level, it needs additional refresh to read new scale and adjust positions properly.

```

Plot(C, "Price", colorBlack, styleLine );

GfxSetOverlayMode(0);

Miny = Status("axisminy");
Maxy = Status("axismaxy");

lvb = Status("lastvisiblebar");
fvb = Status("firstvisiblebar");

pxwidth = Status("pxwidth");
pxheight = Status("pxheight");

TotalBars = Lvb - fvb;

axisarea = 56; // may need adjustment if you are using non-default font for axis

GfxSelectSolidBrush( colorRed );
GfxSelectPen( colorRed );
for( i = 0; i < TotalBars AND i < ( BarCount - fvb ); i++ )
{
    x = 5 + i * (pxwidth - axisarea - 10) / ( TotalBars + 1 );

    y = 5 + ( C[ i + fvb ] - Miny ) * ( pxheight - 10 ) / ( Maxy - Miny );

    GfxRectangle( x - 1, pxheight - y - 1, x + 2, pxheight - y + 2);
}

```

# What's new in latest version?

## Highlights of version 5.00

- New **Watchlist system** featuring:
  - ◆ unlimited number of watch lists
  - ◆ lists keep original order in which symbols were added (still can be sorted alphabetically on-demand)
  - ◆ new **AFL function to refer to watch lists by name**
- Support for **AFL Code Wizard** – brand new automatic formula creation program for people without any programming experience. For more information about AFL Code wizard see this introductory video: <http://www.amibroker.com/video/amiwiz/AFLWiz1.html>
- AFL engine enhancements
  - ◆ new flow control statements: **switch / case / break / continue**
  - ◆ new compound assignment operators: **+=, -=, \*=, /=, %=, &=, |=**
  - ◆ new functions: **GetPlaybackDateTime()**, **PopupWindow()**, Mersene Twister Random Number Generator **mtRandom()**, and others
- New dedicated memory heap allocators for quotes and trading system signals resulting in ability to run much longer optimizations than ever without getting out-of-memory messages
- Two new backtester modes (available using **SetBacktestMode** function) allowing **handling of unfiltered (raw) entry signals**
- User-definable **5-tier commission schedule** in the backtest (Automatic Analysis / Settings)
- Chart template sharing  
now you can **save the chart as "Chart Template, Complete (\*.chart)"** that stores all layout AND referenced formulas in SINGLE file that can be sent to your friend and entire chart will be restored on any computer with ease, without need to copy individual formulas.
- New-Look charts – divider lines between panes are now single pixel and no borders around charts giving cleaner, larger and more readable chart display and printout
- Custom Range Bars (supported in the charts and via **TimeFrameSet()**)
- New **Low-level graphics interface (23 new AFL functions)**
- HTML Import in Automatic Analysis
- Full screen Anti-Aliasing in 3D optimization chart viewer (beautifully smooth 3D charts and improved readability)
- Enhanced Real-Time Quote window display (faster updates, dual-color change marks)
- Control of **Time Shift in the ASCII importer**

## Highlights of version 4.90:

- new **Fundamental data support** including
  - ◆ automatic download from free Yahoo Finance site
  - ◆ access to fundamental data from AFL level
  - ◆ new fundamental data fields in the Information window
- new **Web Research** window
  - ◆ user-definable sites
  - ◆ multiple on-line research windows open simultaneously
  - ◆ flexible auto-synchronization options
- new **Account Manager**
  - ◆ tracking history of all transactions
  - ◆ tracking open position unrealized profit
  - ◆ tracking account equity history
  - ◆ short and long trades, automatic handling of scaling in/out

- ◆ unlimited number of accounts
- ◆ per-account settings/commissions
- new **Bar Replay** tool – great learning tool, featuring
  - ◆ re-playing all symbols' data at once
  - ◆ fast scrolling
  - ◆ user-definable playback speed and interval
- added **Text-To-Speech** capability via **Say()** AFL function. Now AmiBroker can speak out loud any text, for example it can say "Buy 100 shares of AAPL at 91". This is controllable from formula level so you can make it to speak depending on market conditions, signals generated from your formula, etc.
- added ability to fill indicator background with gradient color – via **SetChartBkGradientFill** AFL function.
- new **Fast Fourier Transform** function
- automatic exploration result sorting from AFL level – via **SetSortColumns** AFL function.
- high-resolution performance timing added via **GetPerformanceCounter**, per-chart timed refreshes via **RequestTimedRefresh**
- **HoldMinDays** / **EarlyExitDays** feature in the **backtester**
- 'Every tick' **chart refresh** capability added (Professional Edition only)
- **MDI Tabs** added as UI enhancement.
- **OLE interface** enhancements/additions

For more information about new features please check **USAGE NOTES ON NEW FEATURES**.

#### CHANGES FOR VERSION 5.00.0 (as compared to 4.99.0)

- Database Purify: Date column type in set correctly now so clicking on column sorts by date instead of by alpha (FC#1130)
- Elapsed/estimated time is updated constantly when running long optimizations (FC#1107)
- Extra characters in the title occurring when using dynamic color with Null values in Plot() statements. Fixed. (FC#1129)
- Fixed saving of new commission table (FC#1122)
- In some cases slider did not allow to reach the upper margin of Param() when step was decimal fraction like 0.1 due to floating point rounding. Now it addressed. (FC#1155)
- Price Chart Style is now saved in a layout/template (FC#1039)
- When Find/Replace dialog is still open while Formula Editor is closed it does not cause exception anymore

#### CHANGES FOR VERSION 4.99.0 (as compared to 4.98.0)

- A warning added when closing the last chart window
- Fixed possible stack overflow when using ASCII importer for files without extension but with . (dot) in a path (directory) name
- TimeFrameMode(3) (range mode) was not functional in 4.98, now it is fixed
- Fixed calculation of profit of leveraged instruments denominated in non-base currency when dynamic fx rate was used
- Private heap allocator implemented for quotes and for trading signals  
This should resolve problem of getting "out of memory" problem that could occur during very long optimizations due to poor handling of virtual memory in Windows.

#### CHANGES FOR VERSION 4.98.0 (as compared to 4.97.0)

- N-tick intervals from View→Intraday menu are enabled again in tick databases (were disabled in 4.97 only)
- AFL: Status() function: 2 new fields added
  - "axisminy" – retrieves the minimum (bottom) value of Y axis (indicators only)
  - "axismaxy" – retrieves the maximum (top) value of Y axis (indicators only)

Example 1:

```
Title = "Axis Min Y = " + Status("axisminy") + "Axis Max Y = " +
Status("axismaxy");
```

Example 2:

```
SetChartOptions( 3, 0, ChartGridMiddle );
axismin = Status("axisminy");
axismax = Status("axismaxy");

Plot( C, "Price", colorBlack, styleBar );

// draw exactly 5 grid lines
for( i = 0; i < 5; i++ )
{
    y = axismin + i * (axismax - axismin)/4;
    PlotGrid( y, colorRed );
}
```

- HTML export / import now store extra information about the kind of file exported so when you export optimization result and re-import it later you will be able to use optimization graph (FC#: 1029)
- HTML import: sometimes when importing large files duplicate rows appeared at the end – now it is fixed (FC#1024)
- Fixed formatting of numeric values above 100000 after HTML import when comma used as thousands separator (FC#: 1029)
- Added protection against using file-system reserved characters (\*?#[:></"\\) in watch list names
- Implemented 5-tier commission schedule table in backtester (AA→Settings: Commission table : DEFINE...) (FC#270)
- Account Manager: commission was not subtracted from equity when scaling-in position that was open in previous amibroker run, now it is fixed
- Fixed problem with "Use only local database for this symbol" being set to "yes" during loading of the database with absent broker.master file

## CHANGES FOR VERSION 4.97.0 (as compared to 4.96.0)

- Range Bar compression implemented now (FC#: 210, 1041,897,284)

Range Bars are price-driven bars, with each bar having a required minimum high-low range. Source data are consolidated into one bar until the range requirement is reached, then a new bar is started. It works best with tick data that have only one price per data point. You can use it with other base time intervals as well, but please note that if single source bar H-L range exceeds desired range, the output will be single bar that has range higher than requested. In other words source bars exceeding desired range won't be splitted into several range bars. For example if you use 1 minute bars and there is \$3 dollar movement and you have selected \$1 range bars it won't be splitted into 3 bars. Instead you will get single bar with original \$3 range. This is so because AB has no idea what happened \*inside\* the bar. Prices could move first downwards and later upwards or opposite or zigzagging several times or making any other pattern inside bar and this information is not available from source bar that only has OHLC prices. Note that range bar compression is not standardised. Some other softwares may attempt to split to several artificial bars when range is exceeded, but we believe it is wrong since it is based on assumptions about price action inside bar that may and usually are wrong.

Range bars can now be selected as custom compression from Tools->Preferences->IntradayRange bars are also available via TimeFrameMode() function

```
TimeFrameMode( 3 ); // turn on range bars
TimeFrameSet( 1.5 ); // set compression to 1.5$ range bar
```

- Mersene Twister MT19937 random number generator added

Two new AFL functions:

mtRandom( seed = Null ) – returns single random number (scalar) in the range [0,1)

mtRandomA( seed = Null ) – returns array of random numbers in the range of [0,1)

seed is random generator seed value. If you don't specify one, the random number generator is automatically initialized with current time as a seed that guarantees unique sequence

Both functions use Mersene Twister mt19973ar-cok algorithm.

(Copyright (C) 1997 – 2002, Makoto Matsumoto and Takuji Nishimura.)

Mersene Twister is vastly superior to C-runtime pseudo-random generator available via Random() function.

It has a period of  $2^{19973}$  = approx  $2.9 \cdot 10^{6012}$  For more information visit:

<http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>

See also:

M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator", ACM Transactions on Modeling and Computer Simulation, Vol. 8, No. 1, January 1998, pp 3--30.

- AFL: PopupWindow() function added (FC#: 840)

SYNTAX:

```
PopupWindow( bodytext, captiontext, timeout = 5, left = -1, top = -1 );
```

bodytext – the string containing the text of the window body

caption – the string containing the text of window caption

timeout – auto-close time in seconds (default 5 seconds)  
 left – top-left corner X co-ordinate (default = -1 –means auto-center)  
 top – top-left corner Y co-ordinate (default = -1 – means auto-center)

Example code:

```
if( ParamTrigger("Display Popup Window", "Press here" ) )
{
    PopupWindow("Current time is: " + Now(), "Alert", 2, 640*mtRandom(),
480*mtRandom());
}
```

- ASCII Importer: \$TIMESHIFT option added (FC# 1094)  
 Added ability to time-shift data at the time of the import.

\$TIMESHIFT shift\_in\_hours

For example:

\$TIMESHIFT 2

will import with date/time stamps shifted 2 hours forward

\$TIMESHIFT -11.5

will import with date/time stamps shifted 11 and half hour backward

Please note that only INTRADAY data (with TIME component provided) are shifted. EOD data (without time field) are unaffected.]

- AFL: GetPlaybackDateTime() (FC#: 837)

```
pt = GetPlaybackDateTime(); // new function to retrieve playback position
date/time,
//returns zero if bar replay is NOT active

if( pt )
{
    Title = "Playback time: " + DateTimeToStr( pt );
}
else
{
    Title = "Bar Replay not active";
}
```

It returns zero if bar replay is not active.

- Fixed lockup when bar replay was attempted on base time interval = tick
- Fixed passing IDispatch parameters to OLE/COM method calls

- Fixed error message typo in SetCustomBacktestProc (FC: 1085) "exit" vs "exist"
- 3D optimization chart viewer: added 4x4 Full Screen Anti-Aliasing for beautifully smooth 3D charts and improved readability  
4x4 Full-Screen Anti-Aliasing (FSAA) feature is available only on graphic cards that support this in hardware (all NVidia GeForce cards, and most recent graphic cards that have 3D acceleration). It is not available for low-end graphic cards sometimes found in the cheapest notebooks.  
If FSAA is not supported by the hardware View->Anti-aliasing menu item will be disabled (grayed).  
If FSAA is supported in hardware then it will be turned on by default.  
Note that animation speed with FSAA turned on may be lower on slower graphic cards. If speed is not acceptable you can turn off anti aliasing by unchecking View->Anti-aliasing menu (or turning off "A" button in the toolbar) of 3D chart viewer (O3G.exe).

### CHANGES FOR VERSION 4.96.0 (as compared to 4.95.0)

- HTML Import to AA result list implemented (FC#579)  
Note that this feature is intended to import only HTML files exported by AmiBroker. The HTML parser used is specifically written for this particular purpose and expects certain layout of HTML file. HTML files saved by other programs can not be imported. Use Automatic Analysis -> File -> Import to access this feature.
- Account manager: added "per-trade" commission setting (FC#: 973)
- Two new backtester modes that do not remove redundant signals (FC#: 105)
  - ◆ RAW MODE: signal-based backtest, redundant (raw) signals are NOT removed, only one position per symbol allowed:
  - ◆ RAW MODE WITH MULTIPLE POSITIONS – signal-based backtest, redundant (raw) signals are NOT removed, MULTIPLE positions per symbol will be open if BUY/SHORT signal is "true" for more than one bar and there are free funds, Sell/Cover exit all open positions on given symbol, Scale-In/Out work on all open positions of given symbol at once.
- AFL: new function SetBacktestMode( mode )  
Sets working mode of the backtester:

```
// default, as in 4.90, regular, signal-based backtest, redundant signals
are removed
```

```
SetBacktestMode( backtestRegular );
```

```
// signal-based backtest, redundant (raw) signals are NOT removed, only one
position per symbol allowed
```

```
SetBacktestMode( backtestRegularRaw );
```

```
// signal-based backtest, redundant (raw) signals are NOT removed,
// MULTIPLE positions per symbol will be open if BUY/SHORT signal is "true"
for more than one bar and there are free funds
// Sell/Cover exit all open positions on given symbol, Scale-In/Out work on
all open positions of given symbol at once.
```

```
SetBacktestMode( backtestRegularRawMulti );
```

```
// rotational trading mode - equivalent of EnableRotationalTrading() call
```



```
SetBacktestMode( backtestRotational );
```

- AA window: custom draw buttons (with arrows) are redrawn on killfocus event to prevent flickering
- Fix: when symbol is deleted from database it is removed from all new watchlists as well
- Attempt to assign value to the element of the array named the same as the function inside the function declaration produces error 33 now instead of crash (FC#: 1011)
- Clicking RMB→Watchlist→Remove inside watch list node was removing symbol from all watch lists. Now it is fixed.
- Fixed cell text truncation that could happen randomly when using HTML export
- Position of compound drawings (retracements/cycles) is updated correctly now when "Properties" dialog was chosen from without selecting the drawing via LMB click
- Stops were reset to default values (from settings) when custom backtest routine was stored in separate file (not embedded or included in the formula). Now it is fixed

#### CHANGES FOR VERSION 4.95.0 (as compared to 4.94.0)

- Added support for AFL Code Wizard (**Analysis→AFL Code Wizard** menu and toolbar button)

**AFL Code Wizard is a new add-on for AmiBroker.** It allows creation of trading system formulas without ANY programming experience. It is available for purchase for \$49 one-time fee. Further upgrades are free. AmiBroker comes with trial version that has all functionality \*including\* export of auto-generated AFL formula to AmiBroker. The only missing feature in trial version of AFL CODE wizard is saving the project in .awz format.

To run it select Analysis→AFL Code Wizard menu.

**AFL Code Wizard instructional video can be found at:**

<http://www.amibroker.com/video/amiwiz/AFLWiz1.html>

- Symbol Tree→Right click→Watch List→Type In works from any node now (FC#991)
- Filter window supports more than 256 watchlists (FC#1006)
- Aliases are properly recognized when adding symbols to new watchlists

#### CHANGES FOR VERSION 4.94.0 (as compared to 4.93.0)

- AFL: 23 new low-level graphic functions allowing Windows GDI-like painting  
Completely new low-level graphic AFL interface allows complete flexibility in creating any kind of user-defined display.

The interface mimics closely Windows GDI API, with same names for most functions for easier use for GDI-experienced programmers. The only differences are:

1. compared to Windows GDI all functions are prefixed with 'Gfx'
2. pen/brush/font creation/selection is simplified to make it easier to use and you don't need to care about deletion of GDI objects
3. Three overlay modes are available so you can mix low-level graphics with regular Plot() statements  
(mode = 0 (default) – overlay low-level graphic on top of charts, mode = 1 – overlay charts on top of low-level graphic, mode = 2 – draw only low-level graphic (no regular charts/grid/titles/etc))

Low-level graphics tutorial: [http://www.amibroker.com/guide/h\\_lowlevelgfx.html](http://www.amibroker.com/guide/h_lowlevelgfx.html)

Available low-level gfx functions (click on the links for detailed explanation):

**GfxMoveTo**( x, y ) <http://www.amibroker.com/f?gfxmoveto>  
**GfxLineTo**( x, y ) <http://www.amibroker.com/f?gfxlineto>  
**GfxSetPixel**( x, y, color ) <http://www.amibroker.com/f?gfxsetpixel>  
**GfxTextOut**( "text", x, y ) <http://www.amibroker.com/f?gfxtextout>  
**GfxSelectPen**( color, width = 1, penstyle = penSolid ) <http://www.amibroker.com/f?gfxselectpen>  
**GfxSelectSolidBrush**( color ) <http://www.amibroker.com/f?gfxselectsolidbrush>  
**GfxSelectFont**( "facename", pointsize, weight = fontNormal, italic = False, underline = False, orientation = 0 ) <http://www.amibroker.com/f?gfxselectfont>  
**GfxRectangle**( x1, y1, x2, y2 ) <http://www.amibroker.com/f?gfxrectangle>  
**GfxRoundRect**( x1, y1, x2, y2, x3, y3 ) <http://www.amibroker.com/f?gfxroundrect>  
**GfxPie**( x1, y1, x2, y2, x3, y3, x4, y4 ) <http://www.amibroker.com/f?gfxpie>  
**GfxEllipse**( x1, y1, x2, y2 ) <http://www.amibroker.com/f?gfxellipse>  
**GfxCircle**( x, y, radius ) <http://www.amibroker.com/f?gfxcircle>  
**GfxChord**( x1, y1, x2, y2, x3, y3, x4, y4 ) <http://www.amibroker.com/f?gfxchord>  
**GfxArc**( x1, y1, x2, y2, x3, y3, x4, y4 ) <http://www.amibroker.com/f?gfxarc>  
**GfxPolygon**( x1, y1, x2, y2, ... ) <http://www.amibroker.com/f?gfxpolygon>  
**GfxPolyline**( x1, y1, x2, y2, ... ) <http://www.amibroker.com/f?gfxpolyline>  
**GfxSetTextColor**( color ) <http://www.amibroker.com/f?gfxsettextcolor>  
**GfxSetTextAlign**( align ) <http://www.amibroker.com/f?gfxsettextalign>  
**GfxSetBkColor**( color ) <http://www.amibroker.com/f?gfxsetbkcolor>  
**GfxSetBkMode**( bkmode ) <http://www.amibroker.com/f?gfxsetbkmode>  
**GfxGradientRect**( x1, y1, x2, y2, fromcolor, tocolor ) <http://www.amibroker.com/f?gfxgradientrect>  
**GfxDrawText**( "text", left, top, right, bottom, format = 0 ) <http://www.amibroker.com/f?gfxdrawtext>  
**GfxSetOverlayMode**( mode = 0 ) <http://www.amibroker.com/f?gfxsetoverlaymode>

- AFL: Two new fields in Status() functions: pxwidth and pxheight ( Status("pxwidth") – gives pixel width of chart window (useful for low level graphics functions that operate on pixels) Status("pxheight") – gives pixel height of chart window
- Better scaling and no more black rectangle on the bottom in images created via Edit->Image->Copy / Export
- "New Look" charts (switchable via Tools->Preferences "Charting") – give cleaner, larger and more readable chart display and printout
- Printing/copy-metafile function uses now Enhanced Metafile format that handles clipping regions (styleClipMinMax)

## CHANGES FOR VERSION 4.93.0 (as compared to 4.92.0)

- Fixed exception that might occur when copying to clipboard very long lines from AA result list
- CategoryGetName() now returns empty string for non-existing categories instead of some uninitialized (random) string
- Fixed exception when trying to apply commentary on blank chart pane
- AFL editor fix: right Alt + z does not trigger 'undo' anymore (allows to enter Polish z (z-dot-above) letter)
- Fixed exception occurring when Symbol Information was edited at the same time while first auto-analysis was run
- Fixed bar replay not functioning when viewing interval was the same as base time interval and set to less than 5 minute, mixed mode was off, and no afterhours/weekend filtering was enabled (FC#: 899)
- Fix: defined FXRate for the very first symbol in the database is used correctly now (FC#: 975)
- When LoadFormula() method of Analysis object is called parameters are reset now (FC#: 958)
- Ability to export Chart in "portable" format so you can distribute chart templates to others and they will be restored together with all linked formulas (FC#: 96)  
This feature is available from RIGHT CLICK on chartTemplate->Save...Template->Load... menus.

In addition to old local template format a new one is added with .chart extension that keeps not only window sizes and formula references (paths) but also formulas themselves, so all you need to do is to save your chart into one file (Chart Template, Complete \*.chart) and copy that file onto different computer and chart will be recreated with all formulas linked to it.

To Save chart into new format do the following:

1. Click with RIGHT MOUSE button over the chart and select Template->Save...
2. In the file dialog, "Files of type" combo select "Chart Template, Complete (\*.chart)"
3. Type the file name and click Save.

To load previously saved complete chart do the following:

1. Click with RIGHT MOUSE button over the chart and select Template->Load...
2. In the file dialog, select previously saved \*.chart file and press "Open"

Note: The procedure AmiBroker does internally is as follows: When you save the chart into new format it saves XML file with:

- a) names of all sheets, panes, their sizes, locations and other settings
- b) paths to all formulas used by all panes
- c) the text of formulas themselves

When you load the chart in new format AmiBroker:

- a) sets up the sheets/panes according to information stored in the file
- b) for each formula stored in the file it checks if the same formula exists already on target computer:
  - if it does not exist – it will create one
  - if it exists and the contents is identical to the formula stored in .chart file it will do nothing
  - if it exists and the contents is different then it will create NEW formula file with \_imported.afl suffix (so old file is not touched) and will reference the pane to the \_imported.afl formula instead.

IMPORTANT NOTE: if you use any #include files AmiBroker will store the contents of include files as well inside chart file and will attempt to recreate them on target machine. Please note that in case of includes it will check if it exists and if it is different. If both conditions are met (different file exists already) it will ask to replace or not. If you choose to replace – it will replace and make backup of existing one with .bak extension. If you are using any files in "standard include files and include them using <> braces, AmiBroker will restore files in target machine standard include folder as well even if the standard include folder path is different on the source machine).

Note also – that this functionality is a bit experimental and pretty complex internally. There may be some bugs even though it was tested on number of different setups. Feedback is welcome. It is intended to be used to port charts between different computers. For storing layouts/templates on local computer you should rather use old formats as they consume much less space (they store only references, not the formulas themselves). One may however use new format for archiving purposes as it keeps formulas and all references in one file that is very convenient for backups.

## CHANGES FOR VERSION 4.92.0 (as compared to 4.91.1)

- Watch list tooltips now show WL ordinal number (index), name and symbol count
- new menu item "WatchLists->Hide Empty watchlists" allows to show/hide empty watchlists  
Note that when you add new watch list AmiBroker will automatically unhide empty watch lists so you can see newly added list.
- AFL: Category\* functions work fine now with more than 256 watch lists
- AFL: new function: CategoryFind() allows to search for category by name

SYNTAX: CategoryFind( "name", category )

RETURNS: number

FUNCTION: It allows to search for category by name. It takes category name and kind as parameters and returns INDEX (ordinal number). For example it allows to find watch list index by name:

```
wlnumber = CategoryFind( "MyWatch List 1", categoryWatchlist );
mysymbols = CategoryGetSymbols( categoryWatchlist, wlnumber );
```

The index (in the above example watch list number) can be later used in functions that need the index (like CategoryGetSymbols).

- AFL: new function: InWatchListName() allows to reference watch list by name  
It is equivalent to InWatchList function except that it takes watch list name as parameter instead of the index.

SYNTAX InWatchListName( "listname" )

RETURNS NUMBER

FUNCTION Checks if the stock belongs to a watch list having name of listname. If yes – the function returns 1 otherwise 0.

EXAMPLE

```
Filter= InWatchListName( "mywatchlist" ) OR InWatchListName(
"mysecondwatchlist" );
```

Note that this function is a bit slower than InWatchList() function.

- Fixed Watch List Export – now exports from selected watch lists (not only from watch list number zero)

If you select only one watch list to export then original symbol order is preserved, multiple watch list export uses alphabetical order to prevent duplicates.

- Fixed Watch List Sorting – now sorts selected watch list (not only watch list number zero)

#### CHANGES FOR VERSION 4.91.1 (as compared to 4.91.0)

- fixed symbol tree refresh after Watchlist->Import

#### CHANGES FOR VERSION 4.91.0 (as compared to 4.90.5)

- AA: new options in the context menu "Replace watch list with the results/selected results"  
This new option empties the watch list before adding results. The order of symbols in the result list is preserved in the watch list.
- Watchlists: context (right click) menu – now there is no WL selection dialog displayed  
If you select the watch list from symbol tree and click with RIGHT mouse button to bring up watch list menu the selected watch list is used automatically and watchlist selector dialog is not displayed.
- Watchlists: redesign – now there is **no limit on number of watch lists** you can use
  - a) watch lists are now stored as text files inside "Watchlists" folder inside database. The folder contains of any number of .tls files with watch lists themselves and index.txt that defines the order of watch lists. You can add your own .tls file (one symbol per line) and AmiBroker will update index.txt automatically (adding any new watch lists at the end)The .TLS files can also be open in AmiQuote.
  - b) watch lists remember the order in which symbols were added, so for example if you sort AA result list in some order and then you "add symbols to watch list" the order will be kept in the watch list.
  - c) you can now Add/Delete watch lists using Symbol->Watch List-> menu or from watch list context menu  
Note that if you have done any customization to the menu, you may need to go to Tools->Customize, select "Menu Bar" and press "Reset" button for this new menu items to appear.
  - d) you can now alphabetically sort the symbols in the watch list

e) all watch lists are shown in the symbol tree now, even if they are empty.

f) for backward compatibility OLE automation WatchListBits/WatchListBits2 properties of Stock object continue to work for first 64 watch lists (bit states are transparently emulated)

g) Watch lists created in 4.91 are not visible for older versions

- When changing selected symbol, the tree is not traversed to the bottom (root). Instead only current branch is checked and if symbol is selected only if it is present under this branch (This prevents unnecessary unfolding of "All" and other branches)
- AFL Editor: AUTOWORDSELECTION turned off
- AFL: switch/case statement added (completed: 2007-03-31) (ext.ID: 580). More information: <http://www.amibroker.com/guide/v50/keyword/switch.html>

Example:

```
for( n = 0; n < 10; n++ )
{
    printf("Current n = %f\n", n );

    switch(n) {
        case 0:
            printf("The number is zero.\n");
            break;
        case 3:
        case 5:
        case 7:
            printf("n is a prime number\n");
            break;
        case 2: printf("n is a prime number\n");
        case 4:
        case 6:
        case 8:
            printf("n is an even number\n");
            break;
        case 1:
        case 9:
            printf("n is a perfect square\n");
            break;
        default:
            printf("Only single-digit numbers are allowed\n");
            break;
    }
}
```

- AFL: break/continue statements added (supported inside for/while/do-while/switch statements). More information: <http://www.amibroker.com/guide/v50/keyword/break.html>  
<http://www.amibroker.com/guide/v50/keyword/continue.html>

Example:

```
for( i = 1; i < 1000; i *= 2 )
{
    if( i > 50 ) break;
    printf( "%f\n", i );
}
```

- AFL: new C-like assignment operators +=, -=, \*=, /=, %=, &=, |=

New operators are shortcuts for some common operations.

x += y; is equivalent to x = x + y;  
 x -= y; is equivalent to x = x - y;  
 x \*= y; is equivalent to x = x \* y;  
 x /= y; is equivalent to x = x / y;  
 x %= y; is equivalent to x = x % y;  
 x &= y; is equivalent to x = x & y; // bitwise and  
 x |= y; is equivalent to x = x | y; // bitwise or

- Enhanced display in Real Time Quote window (dual-step change coloring – when field changes it is highlighted with bright yellow for 0.25 second then color changes to pale yellow and after a while to default (white) background))

#### CHANGES FOR VERSION 4.90.0 (as compared to 4.89.0)

- Fixed problem with information window length constraint being set wrong sometimes. (for example TickSize in View->Information was limited to 4 digits after entering Currency field)
- Bar Replay current position static is now updated when changing start date
- Fixed problem RTQ window stopped updating the symbol when it was listed twice in different tabs and removed from one (FC#: 439)
- Fixed data tooltip display when xshift != 0 (FC#: 465)
- Fixed vertical grid Z-order when graphgridzorder = 1 (FC#: 377)
- Added "Use as default" box to Horizontal Line properties dialog – when checked, current drawing "show value" selection is stored as a default for all newly created horizontal lines (FC#: 389)
- Added File->New->Pane (creates blank pane)
- Added File->New->Blank chart (creates blank chart – you can drag-drop formulas on it)
- Window->New Linked window menu has been replaced by File->New->Linked Chart
- #30339 the limit of drawings per symbol has been increased from 5000 to 65000  
 [And contrary to previous (faulty) behaviour when limit is reached, you don't lose your previous drawings. Now you just can't save more than 65000 drawings per one symbol.]

- Renamed Tools->Auto-update quotes (US & Canada) to Auto-update quotes (AmiQuote only) – new installations only
- Removed Tools->Export to CSV – new installations only
- Clipboard copy/paste now works OK on "Symbol ticker" combo (in the toolbar)
- Warning message when deleting symbols changed to "Are you sure to delete current symbol from the DATABASE !"
- Added open interest as a predefined formula
- Account manager: 5-tier commission table added
- Duplicate shortcut Alt+d for Edit / Define removed. Define now uses 'f' shortcut (FC#: 219)
- BarReplay is now paused temporarily when user is drawing objects or printing
- Thick dotted line is now plotted with geometric pen on Win2000/XP or higher (FC#: 720)
- Fixed problem with passing HGDI handles when printing
- Account manager cash calculation when scaling in fixed (FC#: 793)
- Added styleNoRescale to Bollinger Bands in Price (all-in-one).afl
- ASCII importer: volume is now NOT to adjusted when ADJCLOSE and CLOSE is used  
[To turn on adjusting (old, pre-4.90 behaviour) you need to add \$SPLITADJVOL 1 to command to ASCII importer definition]
- FuturesMode setting set using SetOption #24503 works the same as in the settings dialog  
[The difference was due to the fact that PointValue set in the Information window was not transferred to the formula when Futures Mode was NOT set in the Settings dialog (further change by SetOption did not transfer it). Now Point Value is transferred always.]

### CHANGES FOR VERSION 4.89.0 (as compared to 4.88.0)

- AFL: Static variables are now case insensitive (FC#: 663)
- AFL: SetSortColumns() function (FC#: 607)  
SetSortColumns( col1, col2, .... )

sets the columns which will be used for sorting.

col1, col2, ... col10 –Column numbers are ONE-based. Positive number means sort ASCENDING, negative number means sort DESCENDING. Upto 10 columns can be specified for multiple-column sort. Each subsequent call to SetSortColumns overwrites previous one.

Examples:



SetSortColumns( 5 ) – sort by 5th column in ascending order

SetSortColumns( -3 ) – sort by 3rd column in descending order

SetSortColumns( 1, -2 );

– sort by 1st column in ascending order and then by second column in descending order (multiple-column sort).

- Web Research: Synchronization with currently selected symbol has now three options:  
 Don't sync – does not synchronize with currently selected symbol  
 Sync active – synchronizes only when web research is active (or becomes active by clicking on it)  
 Sync always – synchronizes web page always, even if web research window is not active – warning: resource intensive
- New Account Manager added (completed: 2006-12-19)  
 Account manager provides ability to track your account portfolio. Account manager functionality provides superset of features offered by old portfolio manager, but still some things are left to do namely: multi-currency handling, more stats (backtest-like), multi-tiered commission schedules etc.
- Vertical selector line has now user-definable color / bold style and it is plotted behind all graphs so it does not affect readability  
 Use Tools->Preferences->Color to adjust selector line color and style
- Charts: by default extra 3% space is added on the top of chart compared to previous versions (4.88 or earlier)
- printf/strformat are now protected against buffer overrun (1024 bytes)
- Scroll-Bar Zoom feature implemented (dragging left or right edge of chart scrollbar resizes it allowing to zoom in/out from the scrollbar) (FC#619)  
 [To turn it OFF use  
 Tools->Preferences "Charting" tab:  
 UNCHECK "Enable Scroll Bar Zoom" box.]
- RequestTimedRefresh() was not always starting up correctly since 4.86 – fixed now.
- Range markers are now green (begin) and red (end) and are plotted with lines with small 'flag-like' rectangle at the end
- When user types non-existing symbol in ticker box, AmiBroker asks whenever it should be added or not  
 This allows very quick adding of new symbols directly from ticker box.  
 Just type symbol and press ENTER – if it does not exist – it will be added and backfilled if you use real time data source
- When adding/deleting symbols, charts displayed keep selected symbol
- Added ability to create composites (using AddToComposite) inside indicator code via new atcFlagEnableInIndicator flag (FC#: 191)
- AA window: now 'Sync chart' box is hidden when result list is maximized

- AFL: Implemented Say("text") function that speaks provided text (Windows XP, on lower-end Windows you need to install Microsoft Speech API, voice settings are in Windows Control Panel)  
Example:

Say("Testing text to speech engine");

### CHANGES FOR VERSION 4.88.0 (as compared to 4.87.1)

- Automatic Analysis window: Added "Sync chart on select" checkbox  
when turned ON, selection of any line from result list automatically synchronizes (displays) relevant chart. Selection can be made not only by mouse but also by keyboard effectively allowing you to scroll through AA result list using key down button and charts will be switching automatically. Also when you press SPACE button, the arrows for selected trade/signal will appear (only after backtest / scan)
- AFL Lexer code optimizations  
Removed strncmp() C runtime calls from lexer and replaced it with hand optimized code, gives on average 15% speed improvement on AFL execution on larger formulas. NB. multi-line comment handling is speeded up this way as much as 7 times.
- Bar Replay feature implemented (**Tools-->Bar Replay**)  
Bar Replay feature plays back data for ALL SYMBOLS at once. It means that data for all symbols will end at specified "playback position". This affects all formulas (no matter if they are used in charts or auto-analysis).
- Added Insert-->Select to Tools-->Customize-->Keyboard command list to allow customers to add keyboard accelerator to this action
- Fib. TIME extensions , "lock position" now works correctly. Also "bold" attribute can be now applied

### CHANGES FOR VERSION 4.87.1 (as compared to 4.87.0)

- fixed symbol tree, right mouse click on the watch list causing exception

### CHANGES FOR VERSION 4.87.0 (as compared to 4.86.0)

- Added "Request data on save" checkbox to File-->Database Settings-->Intraday Settings  
When this is turned ON, AmiBroker will query data plugin for fresh data when saving database. This ensures that all collected real time data is transferred to AmiBroker's own database.  
Caveats:  
1. if external data source uses automatic backfill – it may cause triggering backfills during saving 2. turning this feature ON may slightly increase CPU load  
Recommendations:  
Generally speaking all data sources with no backfill or slow backfill should turn ON this feature, others (with fast backfill covering long histories) should have this turned OFF.  
IB plugin: recommended setting: ON  
This feature is designed specifically with IB plugin in mind as it has very limited backfill capabilities and it is good to save data so it does not need to be re-filled on next session.  
eSignal plugin: recommended setting: OFF  
Using eSignal with this feature turned on may cause excessive backfills and exceeding eSignal

symbol limit  
 IQFeed plugin: recommended setting: OFF  
 DDE plugin: recommended setting: ON

- Fixed minor save on exit problem
- New menu options
  1. File→New → Default Chart replaces Window→New
  2. File→New → Web Browser – creates new web browser window tab
  3. File→New → Account – creates new account manager window (still in development, not available now) (Tools→Portfolio manager will soon become obsolete)
  4. File→New → Database – replaces File→ New database
  5. File→Save / Save As.– allows to save current document (such as HTML page from the browser, or account information (in development))
  6. File→Save All – saves ALL modified, open documents including all open formula files
  7. File→Close – closes currently open document (chart, browser or account)

Removed View→Profile menu option as it is now replaced by superior File→New→Web Research

- Added Web Research window: multi-tab web browser with multiple built-in financial sites templated shortcuts

This feature effectively replaces and extends previous View→Profile functionality that allowed to view single symbol profile web page. Now via File→New → Web Research you can open multiple (unlimited) tabbed web browsers that allow to display unlimited (user configurable in Tools→Customize→Web Pages. In addition to that the browser now features regular "Address" bar that allows to type ANY URL address and use it as regular web browser.

" Open in New Window" option available from right click menu now opens new web browser inside tab.

" Sync" button when activated synchronizes web page with currently selected symbol, so you can browse for example profiles of different symbols by simply switching active symbol from the tree or combo box, without having to type complex URL addresses.

- Added Customizing of Web Research addresses in Tools→Customize→Web Pages dialog  
 Configuration data are stored in webpages.cfg plain text file that holds any number of URL templates in the form of:

URLTemplate|Description

Description part is optional. URL template may use {t} marker that will be replaced in runtime with currently selected ticker. For more info see: [http://www.amibroker.com/guide/h\\_profiles.html](http://www.amibroker.com/guide/h_profiles.html) (note that old profile window is no longer available however, the URL template encoding remains the same)

- Added ability to turn off MDI tabs (Tools→Customize→Appearance "Show MDI tabs") – note however that it is not recommended to turn them off since window navigation will be more difficult
- Added Symbol→Next Sibling / Symbol→Prev Sibling commands to Tools→Customize→Keyboard page (FC#.ID: 550)  
 This allows to navigate through symbols WITHIN selected category. On NEW installations it will default to Alt+Shift+Left arrow/Right arrow
- Fixed "Remove" button positioning in Organize assignments dialog

- Fixed combo box contents when "Indexes" are chosen in Assignment organizer
- Implemented File→Save All – which saves all open, MODIFIED documents/files, including AFL formulas  
Note that because there is no editing functionality for HTML pages (Web Research window) they are never in modified state, so they never get saved using "Save All". To save HTML page you need to manually do Save on selected HTML doc.
- Pane Up/Pane Down function does not change the original height of panes moved

## CHANGES FOR VERSION 4.86.0 (as compared to 4.85.0)

- Added 'every tick' RT chart refresh in Professional Edition to already existing 1–sec and up refresh intervals  
To enable 'every tick' chart refresh in Professional Edition, go to Tools→Preferences, Intraday tab and enter ZERO (0) into "Intraday chart refresh interval" field. (note Standard Edition won't allow to do that).

Once you enter zero, AmiBroker will refresh all charts with every new trade arriving provided that the formulas you use execute fast enough. If not, it will dynamically adjust refresh rate to maintain maximum possible refresh rate without consuming more than 50% of CPU (on average). So for example if your charts take 0.2 sec to execute AmiBroker will refresh them on average 2.5 times per second.

Note: built-in Windows Performance chart shows cumulated CPU consumption for all processes, to display PER–PROCESS CPU load use SysInternals free software  
<http://www.sysinternals.com/Utilities/ProcessExplorer.html>

- MDI tabs implemented
- AFL: Added GetPerformanceCounter( bReset = False ) function

SYNTAX: GetPerformanceCounter( bReset = False )

RETURNS: Number

GetPerformanceCounter retrieves the current value of the high–resolution performance counter. Returned value is in milliseconds. Resolution is upto 0.001 ms (1 microsecond). The value of high–resolution counter represents number of milliseconds from either system start (boot) or from last counter reset. To reset the counter you need to call GetPerformanceCounter function with bReset parameter set to True. Note that resetting counters inside one formula does not affect counters in other formulas. Since returned values are very large (time in milliseconds since system start is usually quite large), for precise measurements of single function or small function block execution times it is strongly recommended to reset counter at the beginning of the block so floating point resolution (7 digits) does not affect the precision of measurement. Example:

```
GetPerformanceCounter( True ); // reset counter to zero
for( i = 0; i < 1000; i++ )
{
    k = sin( i );
}

elapsed=GetPerformanceCounter();
```

```
"Time [ms] = "+elapsed;
```

The code above shows that 1000 iterations of sin() calculation takes about 1.7 milliseconds. Note that call to the GetPerformanceCounter() has overhead of about 0.015 ms (15 microseconds)

GetPerformanceCounter function may also be used to report time since system start:

```
elapsed=GetPerformanceCounter();
```

```
StrFormat("Time since system start %.0f days, %.0f hours, %.0f minutes,
%.0f seconds, %.0f milliseconds ",
floor(elapsed/(24*60*60*1000)),
floor( elapsed/(60*60*1000) ) % 24,
floor( elapsed/(60*1000) ) % 60,
floor( elapsed/1000 ) % 60,
elapsed % 1000 );
```

It can be also used in trading system automation to measure time in milliseconds between various events (just subtract values returned by GetPerformanceCounter() during two different events)

Caveat: this function relies on Windows API QueryPerformanceCounter function and CPU RTDSC instruction and it may yield to inaccurate results if you have multiple-core processor and AMD's "Cool and Quiet" enabled in BIOS or other CPU clock stepping technologies enabled. If this applies to you, you may check Microsoft hotfix to this problem at: <http://support.microsoft.com/?id=896256>

- Fixed crash that sometimes occurred when chart using RequestTimedRefresh() was closed
- Added "Cancel" button to "Do you want to save database: Yes/No" dialog that appears on AB exit when database is modified
- Added registry key for controlling built-in exception handling.

If you are 3rd party developer working with Visual C++ debugger and wanting exceptions to be caught by Visual Studio debugger instead of AmiBroker's built-in crash recovery system contact AmiBroker support and we will provide instructions how to disable it.

- AFL: atan2( y, x ) implemented

atan2 returns the arctangent of y/x. If x is 0, atan returns 0. If both parameters of atan2 are 0, the function returns 0. While atan returns a value in the range  $PI/2$  to  $PI/2$  radians; atan2 returns a value in the range  $PI$  to  $PI$  radians, using the signs of both parameters to determine the quadrant of the return value.

- AFL: FFT( array, size = 0 ) function implemented

```
FFT( array, len = 0 )
```

performs FFT (Fast Fourier Transform) on last 'len' bars of the array, if len is set to zero, then FFT is performed on entire array. len parameter must be even.

Result:

function returns array which holds FFT bins for first 'len' bars. There are len/2 FFT complex bins returned, where bin is a pair of numbers (complex number): first is real part of the complex number and second

number  
is the imaginary part of the complex number.

```
result = FFT( array, 256 );
```

where:

0th bin (result[0] and result[1]) represents DC component,  
1st bin (result[1 ] and result[2]) represents real and imaginary parts of lowest frequency range  
and so on upto result[ len – 2 ] and result[ len – 1 ]

remaining elements of the array are set to zero.

FFT bins are complex numbers and do not represent real amplitude and phase. To obtain amplitude and phase from bins you need to convert inside the formula. The following code snippet does that:

```
ffc = FFT(data,Len);
for( i = 0; i < Len - 1; i = i + 2 )
{
    amp[ i ] = amp[ i + 1 ] = sqrt(ffc[ i ]^2 + ffc[ i + 1 ]^2);
    phase[ i ] = phase[ i + 1 ] = atan2( ffc[ i + 1 ], ffc[ i ] );
}
```

IMPORTANT note: input array for FFT must NOT contain any Null values. Use Nz() function to convert Nulls to zeros

if you are not sure that input array is free from nulls.

Example formula:

```
SetBarsRequired(100000,100000);

Len = Param("FFT Length", 1024, 64, 10000, 10 );

Len = Min( Len, BarCount );

x = BarIndex();
x1 = x - BarCount + Len;

input = C;
a = LastValue( LinRegIntercept( input, Len - 1 ) );
b = LastValue( LinRegSlope( input, Len - 1 ) );

Lr = a + b * x1;

data = input - Lr;// de-trending

ffc = FFT(data,Len);

for( i = 0; i < Len - 1; i = i + 2 )
{
```

```

    amp[ i ] = amp[ i + 1 ] = sqrt(ffc[ i ]^ 2 + ffc[ i + 1 ]^2);
    phase[ i ] = phase[ i + 1 ] = atan2( ffc[ i + 1 ], ffc[ i ] );
}

auto = ParamToggle("Auto dominant cycle", "No|Yes", 1 );
sbar = Param( "Which FFT bin", 1, 0, 50 );

skipbin1 = ParamToggle("Skip 1st FFT bin", "No|Yes", 1 );

if( auto )
{
    sbar = int( LastValue(ValueWhen( amp == LastValue(Highest( IIf( skipbin1
AND x < 4, 0 , amp ) )), x / 2 )) );
}

fv = Status("firstvisiblebar");

thisbar = Ref( int(x/2) == sbar, -fv);
Plot( Ref(amp,-fv),
"amplitude (bin " + Ref( int(x/2), -fv ) + ")", IIf( thisbar, colorRed,
colorBlack ),styleArea);

Plot( IIf( BarCount - BarIndex() < Len, data, Null ) ,
"de-trended input (" + Len + " bars)", colorOrange, styleLeftAxisScale );
Plot( cos( phase[ sbar * 2 ] + (sbar) * x1 * 2 * 3.1415926 / Len ),
" dominant cycle " + Len/(sbar) + "(" + sbar + " bin) bars", colorBlue,
styleOwnScale );

GraphZOrder=1;
GraphXSpace = 10;

```

- Fixed: pre-defined color table in broker.prefs sometimes got corrupted
- Repainting of RT quote window is now flicker-free on Windows XP and higher
- Symbol->Organize assignments, added ability to remove multiple symbols at once from watch list, index and favourite categories

## CHANGES FOR VERSION 4.85.0 (as compared to 4.84.0)

- AFL: Added DateTimeConvert function (FC#: 297)  
**DateTimeConvert**( format, date, time = Null )

The function allows to convert from DateTime format to DateNum and TimeNum and vice versa.

format parameter controls the direction of conversion:

format = 0 – converts DateTime format to DateNum format, example

```
mydatenum = DateTimeConvert( 0, DateTime() );// - this returns DateNum
```

date argument should be in datetime format time argument in this case should not be used

format = 1 – converts DateTime format to TimeNum format, example:

```
mytimenum = DateTimeConvert( 1, DateTime() ); // - returns timenum
```

date argument should be in datetime format time argument in this case should not be used

format = 2 – converts from DateNum and optionally TimeNum to DateTime format, example:

```
mydatetime = DateTimeConvert( 2, DateNum(), TimeNum() );
```

date argument should be in datenum format time argument (optional) should be in timenum format. In case of EOD data you can skip time argument:

```
mydatetime = DateTimeConvert( 2, DateNum() );
```

- K-ratio is now reported with 4 decimal places (FC#: 357)
- Added 'left side' display option to horizontal line labels and fixed problem with label sometimes disappearing (completed: 2006-08-15) (FC#: 342)
- AFL: **fgetstatus**( filename, what, format = 0 )  
Implemented

**fgetstatus**( filename, what, format = 0 )

function that retrieves file properties/status.

Returns NUMBER or STRING depending on format parameter  
If file does not exist it returns Null.

Parameters:

filename – the name of the file (with or without full path) to query

what – specifies what file property to retrieve, allowable values

0 – the date/time the file was created

1 – the date/time the file was last modified

2 – the date/time the file was last accessed for reading

3 – the file size in bytes

4 – attribute byte of the file

format – specifies return format of date/time values (format specifications are the same as in Now() function):

allowed values:

0 – returns string containing date/time formatted according to system settings

1 – returns string containing date only formatted according to system settings

2 – returns string containing time only formatted according to system settings

3 – returns DATENUM number with date

4 – returns TIMENUM number with time

5 – returns DATETIME number with date/time

6 – returns date DAY (1..31)

7 – returns date MONTH (1..12)

8 – returns date YEAR (four digit)



- 9 – returns date DAY OF WEEK (1..7, where 1=Sunday, 2=Monday, and so on)
- 10 – returns date DAY OF YEAR (1..366)

Note that Windows supports only 2 second resolution of file date/time stamps.

Example:

```
// get modification date string of portfolio.afl file
fgetstatus( "formulas\\Equity\\portfolio.afl", 1, 0 );
```

- AFL: GetTradingInterface uses WaitForInputIdle now instead of fixed interval wait when launching IB interface
- Disabled snapping to pixel resolution after manual editing of drawings #32071

### CHANGES FOR VERSION 4.84.0 (as compared to 4.83.1)

- Added GradientFillSlow code to support gradients on printers and metafiles which do not support them natively (completed: 2006–07–28)
- Added GraphGridZOrder variable to control when grid lines are plotted (completed: 2006–07–31) (FC#: 351)  
When you specify  
GraphGridZOrder = 1;  
then grid lines are plotted on top of any charts
- AFL: new function RequestTimedRefresh( interval, onlyvisible = True ) (completed: 2006–08–06)

**RequestTimedRefresh**( interval, onlyvisible = True )

– causes given indicator window to refresh automatically every <interval> seconds regardless of data source used or connection state.

**interval** parameter defines timeout in seconds between refreshes. AmiBroker attempts to align refreshes to second boundary so if you call it RequestTimedRefresh( 5 ) you should get refreshes at 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50 and 55 second of the minute. Due to the way how regular (low overhead) timers are implemented in Windows they have accuracy of +/-55ms provided that CPU is not very busy. Don't expect to get first line of your code to execute exactly at .000 milliseconds. This varies depending on machine load, number of quotes, system time slice and tens of other factors. Usually (on my testing machines) the first line of the code executes anywhere in the first 100 ms of the second, provided that other processes do not interfere. Windows is not real-time operating system and it does not guarantee any fixed execution/reaction times.

**onlyvisible** parameter set to True (default value) means that refreshes are triggered only for visible and not minimised windows. This applies also to main AmiBroker window – when it is minimised charts are NOT refreshed by default. To force refreshes when window is minimised you need to set this parameter to False. Note that this visibility applies to mostly to 'minimised' state or the situation when you move chart outside the boundary of physical screen so it is not visible to an eye but still open. It does not apply to chart windows that are on placed on inactive sheets, as they do not really exist until they are shown (this way AmiBroker conserves memory and CPU) and as non-existing, can not be refreshed.

Example:

`RequestTimedRefresh( 5 );` // automatically refresh this particular chart every 5 seconds

Hint: to detect whenever given refresh comes from timer or user action you can use `Status("redrawaction")` function. It returns 0 for regular refresh (user action) and 1 for timer-refresh

- AFL: `Status("redrawaction")` added (completed: 2006-08-06)  
Status function now supports new field: redrawaction

```
Status( "redrawaction" )
```

It returns 0 (zero) for regular refreshes, and 1 for refreshes triggered via `RequestTimedRefresh()`. Note that in the future more return values can be added.

Example:

```
if( Status( "redrawaction" ) ==1 )
{
_TRACE( "\nTIMED REFRESH"+Now( ) );
}
RequestTimedRefresh( 1 );
```

- `CreateStaticObject()` actually calls internal `CreateStaticObject :-)` – for some reason it was calling regular `CreateObject()` instead. (completed: 2006-07-27)
- Fixed problem with `HoldMinDays` sometimes allowing exit too soon (completed: 2006-07-29)
- Fixed: Custom metric sort problem in optimization result list (occurred in 4.83 only) (completed: 2006-07-29) (FC#: 345)
- Pane resizing algorithm changed so if you drag the divider between panes only neighbouring panes are resized (others remain unchanged)
- Real time quote parent window has `WS_CLIPCHILDREN` style in order to fix some minor redraw issue (completed: 2006-07-31)

#### CHANGES FOR VERSION 4.83.1 (as compared to 4.83.0)

- Fix to FC#302 caused problems with calling COM properties with parameters. This is now fixed.

#### CHANGES FOR VERSION 4.83.0 (as compared to 4.82.0)

- Added detection of multiple copies of plugins for the same data source (completed: 2006-07-26)  
This (multiple copies of the plugins for same data source) was causing problems and lots of support issues because people were using old plugins even if they installed (newer) copy of the plugin. This was so because AB always used first one encountered in given directory and it usually was the oldest one. Now AB displays warning message that there are multiple copies of the same plugin and then they must be renamed or removed.
- Added `HoldMinDays` and `EarlyExitDays` options to `SetOption()` (completed: 2006-07-27)  
In addition to already existing functionality of `HoldMinBars` / `EarlyExitBars`, new options to specify min hold/penalty period in CALENDAR DAYS (as opposed to bars).

```
SetOption("HoldMinDays", 30 ); // set minimum holding period to 30 calendar
days
SetOption("EarlyExitDays", 90 ); // set penalty period to 90 calendar days
```

Note that you can use one or another (not both at the same time) for one parameter, so you can specify minimum holding period in either days or bars (but not both at the same time). The same with EarlyExit period. You may however use different specifications (i.e. days/bars) for EarlyExit and HoldMin (they are independent from each other).

- Added **scoreExitAll** constant that causes rotational mode backtester to exit all positions regardless of HoldMinBars (completed: 2006–07–27)

Note that this is global flag and it is enough to set it for just any single symbol to exit all currently open positions, no matter on which symbol you use scoreExitAll (it may be even on symbol that is not currently held). By setting PositionScore to scoreExitAll you exit all positions immediately regardless of HoldMinBars setting.

Example:

```
PositionScore = ..your normal score..
// if market timing signal goes negative close all positiions
PositionScore = IIf( MarketTimingSignal < 0, scoreExitAll, PositionScore );
```

NOTE: This flag works ONLY in rotational trading mode (in regular mode it is easy to generate such global exit by ORing sell signal with global sell signal).

- CBT: Added **ExitReason** property to Trade object (completed: 2006–07–18) (FC#: 88)
- Chart Tree – RMB – Edit – now restores and focuses to already opened formula editor (if it is during editing) (completed: 2006–07–24) (FC#: 300)  
Note that you can still open NEW instance of the editor with the very same file (for comparison and to see the original contents) by holding down SHIFT key when choosing "EDIT" menu.
- Eliminated extra unnecessary refreshes of commentary window when new symbol was picked from ticker selector (completed: 2006–07–24)
- Fixed calling parameter-less JScript functions from AFL level via GetScriptObject() (completed: 2006–07–19) (FC #: 302)
- Fixed: wrong estimated time in progress dialog when number of steps is greater than  $2^{32}$  (4294967296) (allocated to: 4.80.2) (completed: 2006–07–24)
- AFL: Added Gradient background coloring (SetChartBkGradientFill function) (completed: 2006–07–26)  
Gradient fill of chart interior is now available via SetChartBkGradientFill function.  
Please note that this is independent from chart background color (background color fills entire pane, gradient fill is only for actual chart interior, so axes area is not affected by gradient fill)

**SetChartBkGradientFill**( topcolor, bottomcolor, titlebkcolor = default )

topcolor – specifies top color of the gradient fill

bottomcolor – specifies bottom color of the gradient fill

titlebkcolor – (optional) the background color of title text. If not specified then top color is automatically used for title background.

Example:

```
SetChartBkGradientFill( ParamColor( "BgTop",
colorWhite), ParamColor( "BgBottom", colorLightYellow) );
```

- Multiple tabs in RTQ window (completed: 2006–07–18) (FC#: 22)  
RT quote window tabs behave the same as chart tabs: can be renamed with right mouse button and dragged from one place to another.  
20 tabs are currently available.
- Non-numeric custom trade metrics are not formatted with thousands separators anymore (completed: 2006–07–21) (FC#: 314)
- now Tools->Preferences->Misc "Decimal places in RT quote window" allows to specify upto 6 decimal places. (completed: 2006–07–25)
- OLE: Fixed default property (Item) marker for Windows collection (completed: 2006–07–18)
- Progress dialog: now estimated time is displayed in years/days if it is large enough (completed: 2006–07–24)

## CHANGES FOR VERSION 4.82.0 (as compared to 4.81.1)

- OLE: Quotations collection does not have \_Item hidden property so it should fix problems with Delphi
- OLE: fixed Documents collection Item method returning always first document object regardless of parameter value
- OLE: Documents collection prepared to support multiple document types
- OLE Added Windows collection, Document property and Activate and Close methods to Window object (FC #272)

Windows collection has one method:

Add – adds new window

and two properties:

Count – returns the number of windows within given document

Item( i ) – returns the i-th window object

Window object has two new methods:

Activate() – makes window active (focused)

Close() – closes given window

and one new property:

Document – gives parent document object

#### EXAMPLE CODE (JScript):

```

outBuffer = ""; // global output string buffer

function output( string )
{
    outBuffer = outBuffer + string + "\n";
}

AB = new ActiveXObject( "Broker.Application" );

Docs = AB.Documents;
DocsQty = Docs.Count;

for( i = 0; i < DocsQty; i++ )
{
    Doc = Docs( i );

    output( "Document " + i + " name : " + Doc.Name );

    Windows = Doc.Windows;
    WinsQty = Windows.Count;

    for( j = 0; j < WinsQty; j++ )
    {
        Win = Windows( j );

        output( " Window " + j + " Selected tab " + Win.SelectedTab );
    }
}

WScript.echo( outBuffer );

WScript.echo( "Now will add one window" );
Doc = Docs( 0 );
Windows = Doc.Windows;

NewWindow = Windows.Add();

WScript.echo( "and close it" );

NewWindow.Close();

```

- New menu Symbol->Favorities and new item "Erase favourites" to empty the list of favourites (FC# 280)
- Horizontal line value label added (use new Study properties -> Show Y Value checkbox available for horizontal lines) (FC# 271)
- Fixed crash when clicking on Watch List using RMB (problem occurred in 4.81 beta) (FC# 243)
- Fix: Status("lastvisiblebarindex") and Status("lastvisblebar") now return zero based index instead of one-based (FC# 265)
- Fix: removed spikes from styleCloud when plotted in non-continuous way (with nulls in the middle of the array) (FC# 283)
- Fix rounding problem in AA in futures mode when difference between entry and exit prices was very small and pointvalue large (added extra rounding to profit calculation) (FC# 135)
- Delete key now works in real time quote window (FC# 266)

- AddTextColumn produces now LEFT-justified output (FC# 291)

### CHANGES FOR VERSION 4.81.1 (as compared to 4.81.0)

- fixed problem with some of the watch lists not showing up in 4.81.0

### CHANGES FOR VERSION 4.81.0 (as compared to 4.80.2)

- Database structure changed to hold new fundamental data – quarterly EAT/EBT/Sales figures, Code and Nominal value have been replaced by new fields.

Despite these changes files are still backward and forward compatible – so it can be read by any other AB version, the only minor inconvenience is that if you load new file into old version (pre-4.81) of AmiBroker you will see weird numbers in old "Finances" dialog as well as in "Shares out." and "Book Value" fields.

- Implemented new View->Information property page featuring extra fundamental data fields
- OLE: added new fundamental data fields to Stock object

Stock object has now the following properties:

```
string Ticker;
Collection Quotations;
string FullName;
boolean Index;
boolean Favourite;
boolean Continuous;
long MarketID;
long GroupID;
float Beta;
float SharesOut;
float BookValuePerShare;
float SharesFloat;
string Address;
string WebID;
string Alias;
boolean IsDirty;
long IndustryID;
long WatchListBits;
long DataSource;
long DataLocalMode;
float PointValue;
float MarginDeposit;
float RoundLotSize;
float TickSize;
long WatchListBits2;
string Currency;
string LastSplitFactor;
DATE LastSplitDate;
float DividendPerShare;
DATE DividendPayDate;
```

```

DATE ExDividendDate;
float PEGRatio;
float ProfitMargin;
float OperatingMargin;
float OneYearTargetPrice;
float ReturnOnAssets;
float ReturnOnEquity;
float QtrlyRevenueGrowth;
float GrossProfitPerShare;
float SalesPerShare;
float EBITDAPerShare;
float QtrlyEarningsGrowth;
float InsiderHoldPercent;
float InstitutionHoldPercent;
float SharesShort;
float SharesShortPrevMonth;
float ForwardDividendPerShare;
float ForwardEPS;
float EPS;
float EPSEstCurrentYear;
float EPSEstNextYear;
float EPSEstNextQuarter;
float OperatingCashFlow;
float LeveredFreeCashFlow;

```

- AFL: new function `GetFnData()` – allows accessing fundamental data

SYNTAX: **GetFnData**("field")

RETURNS: number

"field" can be one of the following:

```

"EPS"
"EPSEstCurrentYear"
"EPSEstNextYear"
"EPSEstNextQuarter"
"PEGRatio"
"SharesFloat"
"SharesOut"
"DividendPayDate"
"ExDividendDate"
"BookValuePerShare"
"DividendPerShare"
"ProfitMargin"
"OperatingMargin"
"OneYearTargetPrice"
"ReturnOnAssets"
"ReturnOnEquity"
"QtrlyRevenueGrowth"
"GrossProfitPerShare"
"SalesPerShare"
"EBITDAPerShare"

```

"QtrlyEarningsGrowth"  
 "InsiderHoldPercent"  
 "InstitutionHoldPercent"  
 "SharesShort"  
 "SharesShortPrevMonth"  
 "ForwardDividendPerShare"  
 "ForwardEPS"  
 "OperatingCashFlow"  
 "LeveredFreeCashFlow"  
 "Beta"  
 "LastSplitRatio"  
 "LastSplitDate"

- ASCII importer: added support for new fundamental data fields :

\$FORMAT command now supports additional fields:

DIV\_PAY\_DATE  
 EX\_DIV\_DATE  
 LAST\_SPLIT\_DATE  
 LAST\_SPLIT\_RATIO  
 EPS  
 EPS\_EST\_CUR\_YEAR  
 EPS\_EST\_NEXT\_YEAR  
 EPS\_EST\_NEXT\_QTR  
 FORWARD\_EPS  
 PEG\_RATIO  
 BOOK\_VALUE (requires SHARES\_OUT to be specified as well)  
 BOOK\_VALUE\_PER\_SHARE  
 EBITDA  
 PRICE\_TO\_SALES (requires CLOSE to be specified as well)  
 PRICE\_TO\_EARNINGS (requires CLOSE to be specified as well)  
 PRICE\_TO\_BV (requires CLOSE to be specified as well)  
 FORWARD\_PE (requires CLOSE to be specified as well)  
 REVENUE  
 SHARES\_SHORT  
 DIVIDEND  
 ONE\_YEAR\_TARGET  
 MARKET\_CAP (requires CLOSE to be specified as well – it is used to calculate shares outstanding)  
 SHARES\_FLOAT  
 SHARES\_OUT  
 PROFIT\_MARGIN  
 OPERATING\_MARGIN  
 RETURN\_ON\_ASSETS  
 RETURN\_ON\_EQUITY  
 QTRLY\_REVENUE\_GROWTH  
 GROSS\_PROFIT  
 QTRLY\_EARNINGS\_GROWTH  
 INSIDER\_HOLD\_PERCENT  
 INSTIT\_HOLD\_PERCENT  
 SHARES\_SHORT\_PREV  
 FORWARD\_DIV



OPERATING\_CASH\_FLOW  
 FREE\_CASH\_FLOW  
 BETA

(see Formats\aqfe.format and Formats\aqfn.format files for example usage)

- Removed Stock→Information and Stock→Finances dialogs (this functionality is replaced by View→Information – new dockable property window)
- AFL: new function StrReplace( string, oldsubstring, newsubstring )
- Fixed crash occurring sometimes if "Symbol→Watch List→Type-in" symbol was longer than maximum allowable length (allocated to: 4.80.2)
- Foreign() function changed. Now by default missing data bar OHL fields are filled using previous bar Close and volume is set to zero.  
 It is possible to turn old behaviour (filling missing bar OHL fields using previous bar OHL fields and copying previous bar volume) if you use  
 Fixup parameter set to 2.  
 Foreign("MSFT","O", 2 ); // old-style (pre-4.81) fill
- Scaling-in profit% calculations modified to use total cost instead of max. pos value
- ships with AmiQuote 1.90 featuring automatic download and import of fundamental data from Yahoo Finance. See [AmiQuote\ReadMe2.html](#) for the details.

#### CHANGES FOR VERSION 4.80.2 (as compared to 4.80.1)

- Changed %profit calculation when scaling-in is used to use maximum number of shares ever held as a base of %profit calculation – this leads to most conservative % figures, compared to using just initial entry value.
- Prec() function improved so it does not show rounding errors when working on integer

#### CHANGES FOR VERSION 4.80.1 (as compared to 4.80.0)

- fixed problem with Easy Alert window not accepting new alerts in some circumstances
- included new IB plugin version 1.6.2 (FC issue #54)

**CHANGES FOR EARLIER VERSIONS ARE DOCUMENTED IN RELEASE NOTES DOCUMENT THAT YOU CAN FIND IN AMIBROKER INSTALLATION FOLDER.**

## Usage notes on new features in 5.00

- New [Watchlist system](#) featuring:
  - ◆ unlimited number of watch lists
  - ◆ lists keep original order in which symbols were added (still can be sorted alphabetically on-demand)
  - ◆ new [AFL function to refer to watch lists by name](#)

- Support for **AFL Code Wizard** – brand new automatic formula creation program for people without any programming experience. For more information about AFL Code wizard see this introductory video: <http://www.amibroker.com/video/amiwiz/AFLWiz1.html>
- AFL engine enhancements
  - ◆ new flow control statements: [switch /case / break / continue](#)
  - ◆ new compound assignment operators: +=, -=, \*=, /=, %=, &=, |=
  - ◆ new functions: [GetPlaybackDateTime\(\)](#), [PopupWindow\(\)](#), Mersene Twister Random Number Generator [mtRandom\(\)](#), and others
- New dedicated memory heap allocators for quotes and trading system signals resulting in ability to run much longer optimizations than ever without getting out-of-memory messages
- Two new backtester modes (available using [SetBacktestMode](#) function) allowing [handling of unfiltered \(raw\) entry signals](#)
- User-definable [5-tier commission schedule](#) in the backtest (Automatic Analysis / Settings)
- Chart template sharing  
now you can [save the chart as "Chart Template, Complete \(\\*.chart\)"](#) that stores all layout AND referenced formulas in SINGLE file that can be sent to your friend and entire chart will be restored on any computer with ease, without need to copy individual formulas.
- New-Look charts – divider lines between panes are now single pixel and no borders around charts giving cleaner, larger and more readable chart display and printout
- Custom Range Bars (supported in the charts and via [TimeFrameSet\(\)](#))
- New [Low-level graphics interface \(23 new AFL functions\)](#)
- HTML Import in Automatic Analysis
- Full screen Anti-Aliasing in 3D optimization chart viewer (beautifully smooth 3D charts and improved readability)
- Enhanced Real-Time Quote window display (faster updates, dual-color change marks)
- Control of [Time Shift in the ASCII importer](#)

## Usage notes on new features in 4.90

- new [Fundamental data support](#) including
  - ◆ automatic download from free Yahoo Finance site
  - ◆ access to fundamental data from AFL level
  - ◆ new fundamental data fields in the Information window
- new [Web Research](#) window
  - ◆ user-definable sites
  - ◆ multiple on-line research windows open simultaneously
  - ◆ flexible auto-synchronization options
- new [Account Manager](#)
  - ◆ tracking history of all transactions
  - ◆ tracking open position unrealized profit
  - ◆ tracking account equity history
  - ◆ short and long trades, automatic handling of scaling in/out
  - ◆ unlimited number of accounts
  - ◆ per-account settings/commissions
- new [Bar Replay](#) tool – great learning tool, featuring
  - ◆ re-playing all symbols' data at once
  - ◆ fast scrolling
  - ◆ user-definable playback speed and interval
- added **Text-To-Speech** capability via [Say\(\)](#) AFL function. Now AmiBroker can speak out loud any text, for example it can say "Buy 100 shares of AAPL at 91". This is controllable from formula level so you can make it to speak depending on market conditions, signals generated from your formula, etc.

- added ability to fill indicator background with gradient color – via [SetChartBkGradientFill](#) AFL function.
- new **Fast Fourier Transform** function
- automatic exploration result sorting from AFL level – via [SetSortColumns](#) AFL function.
- high–resolution performance timing added via [GetPerformanceCounter](#), per–chart timed refreshes via [RequestTimedRefresh](#)
- **HoldMinDays** / **EarlyExitDays** feature in the [backtester](#)
- 'Every tick' [chart refresh](#) capability added (Professional Edition only)
- **MDI Tabs** added as UI enhancement.
- [OLE interface](#) enhancements/additions

## Usage notes on new features in 4.80

- **Brand new fully customizable User Interface**
  - ◆ advanced nested docking
  - ◆ sliding auto–hide panes
  - ◆ tear–off tabs
  - ◆ advanced customizable toolbars and menus
  - ◆ themed appearance
- **much better performance**
  - ◆ upto 480% faster arithmetic and logical operator array processing
  - ◆ faster database handling
  - ◆ startup time decreased 10 times
- **charting improvements**
  - ◆ new chart styles
  - ◆ 16 million color support
  - ◆ x–shift feature
  - ◆ [PlotText](#) function
- **other improvements / new features**
  - ◆ multiple–column sorting in Automatic analysis
  - ◆ N–volume bar compression
  - ◆ OLE automation new features (exporting images, loading/saving templates, zooming)
  - ◆ new plugins (IB plugin with 30+ day backfill, more stable IQFeed plugin)
  - ◆ [easy real–time alerts](#)
  - ◆ bid/ask/trade filtering in time&sales window
  - ◆ ability to import tick data added to ASCII importer
  - ◆ handling of HoldMinBars/EarlyExitFee in the backtester
- **64–bit version available (World's first 64–bit TA program)**
  - ◆ +25% faster execution (compared to 32 bit version running on the same hardware)
  - ◆ huge addressable memory (currently upto 1024 GB RAM)
  - ◆ compatible with newest x64 Windows versions

## Usage notes on new features in 4.70

***A few most important new features and changes (click on the link to find more):***

- **Drag–and–drop charting interface**

Now AmiBroker makes creation of indicators a breeze. Just drag–and–drop one indicator onto another to create completely new indicator without writing single line of code.

[Click here to learn more](#)

- **Support for Pyramiding / Scaling and multiple currencies in portfolio backtester**

Now system traders can backtest advanced strategies that scale-in and scale-out trades depending on market conditions. You can also backtest multiple symbols that trade in different base currencies.

[Click here to learn more](#)

- **Context-sensitive on-line help**

Press F1 key anywhere in the program (including pressing F1 when any menu is displayed) to see relevant help page displayed automatically

- **Advanced portfolio backtester interface**

Get full control over portfolio backtester with new advanced interface. Possibilities are endless.

[Click here to learn more](#)

- **User-definable backtest metrics**

Now you can extend built-in backtest reports to include metrics that you invent. New metrics appear also in the optimization result list allowing you to plot your custom statistics against optimization variables in beautiful animated 3D charts.

[Click here to learn more](#)

- **time-and-sales window (RT)**

Time and sales window provides tick-by-tick insight into market activity

[Click here to learn more](#)

- **automatic trading interface (for Interactive Brokers)**

Are you bored with placing orders manually ? Now you can automate placing orders through our automatic trading interface for Interactive Brokers.

Automatic trading interface is available for FREE in a separate download from <http://www.amibroker.com/download.html>

- **Time-frame functions supporting N-day and N-tick intervals**

Now you can use `TimeFrameSet` to access intervals like 3-day, 17-tick data from AFL formula level

- **new AFL editor**

New editor allows editing multiple files at once and can be expanded to cover full screen for more comfortable code editing

- **improved syntax checking and error reporting**

AFL engine error reporting is much improved: error line is highlighted with yellow color background and you get full explanation of an error with coding example when you press F1 key in the AFL editor when error is highlighted.

- **new AFL functions**

new functions allow more control over chart parameters, additional file operations (creating / removing

files and directories) and other useful stuff

## Usage notes on new features in 4.60

- **Three-dimensional animated optimization charts**

Find the most robust parameter set for your trading system using stunning 3D animated surface charts.

[Click here to LEARN MORE...](#)

- **3 brand-new drawing tools: Fibonacci Extension, Fibonacci Time, Cycles**

New drawing tools allow discretionary traders to better project major market moves.

[Click here to LEARN MORE...](#)

- **Relative Performance chart**

Great tool to compare the performance of many different symbols with ease.

[Click here to LEARN MORE](#)

- **Notepad window**

Notepad window is a great tool to store free-text notes about particular security. Just type any text and it will be automatically saved / read back as you browse through symbols. Notes are global and are saved in "Notes" subfolder as ordinary text files. Notes can be also read and written to via AFL NoteGet and NoteSet functions.

[Click here to learn more...](#)

- **Mixed EOD/Intraday mode (allowing long EOD histories and intraday data in the same database)**

Now you can keep both long EOD and intraday data in single database. New eSignal provides now 3+ years of daily history and 120 days of intraday data (1/2 a year) in new mixed mode.

[Click here to LEARN MORE...](#)

- **new real-time data plugins for Interactive Brokers and DDE sources**

lower your real-time data fees using free data from your broker. Learn more about [Interactive Brokers plugin](#) and [DDE plugin](#).

- **Day and night trading session handling with individual settings**

Now intraday activity can be filtered so only day, night, or both trading sessions are shown. You can also watch entire 24 hours activity. And all this is configurable on per-group level so various symbols can have various trading start/end times.

[Click here to LEARN MORE...](#)

- **browser-like history Forward/Back buttons**

small but usefull feature – now you can click "BACK" button like in Internet Explorer to go to the previously watched symbol. History is kept individually for each chart window and tracks all your moves.

- **15 new AFL functions:**

clipboard access: [ClipboardGet](#), [ClipboardSet](#),  
 reading automatic-analysis settings: [GetOption](#),  
 access to streaming RT data such as bid/ask: [GetRTData](#),  
 get quotation mode: [IsContinuous](#),  
 read/write notes from AFL level: [NoteGet](#), [NoteSet](#),  
 new parameter types: [ParamDate](#), [ParamTime](#),  
 static variables: [StaticVarGet](#), [StaticVarGetText](#), [StaticVarSet](#), [StaticVarSetText](#),  
 dynamic variables: [VarGet](#), [VarSet](#)

- **OLE automation interface improvements**

Now you can control portfolio backtester from external script, plus you have control over Commentary window [Click here to learn more...](#)

- **new data sources in AmiQuote: Forex and Integratir**

## Usage notes on new features in 4.50

*A few most important new features and changes (click on the link to find more):*

- **PORTFOLIO – LEVEL BACKTESTING and OPTIMIZATION**

New backtester **works on PORTFOLIO LEVEL**, it means that there is single portfolio equity and position sizing refers to portfolio equity. Portfolio equity is equal to available cash plus sum of all simultaneously open positions at given time.

AmiBroker's **portfolio backtester** lets you combine trading signals and trade sizing strategies into simulations which exactly mimic the way you would trade in real time. A core feature is its ability to perform dynamic money management and risk control at the portfolio level. Position sizes are determined with full knowledge of what's going on at the portfolio level at the moment the sizing decision is made. Just like you do in reality.

[Click here to LEARN MORE...](#)

- **MULTIPLE TIME-FRAME support in AFL**

Allows you to combine charts, signals, etc from multiple time frames in single formula.

[Click here to LEARN MORE...](#)

- **New statistics and metrics in backtest reports**

[Click here to LEARN MORE...](#)

- **New RESULT EXPLORER application for browsing/viewing/organizing backtest results**

- **OTHER NEW AFL FUNCTIONS**

- ◆ **CategoryAddSymbol** – adds a symbol to a category (AFL 2.5)
- ◆ **CategoryGetName** – get the name of a category (AFL 2.5)
- ◆ **CategoryGetSymbols** – retrieves comma-separated list of symbols belonging to given category (AFL 2.5)
- ◆ **CategoryRemoveSymbol** – remove a symbol from a category (AFL 2.5)
- ◆ **IsFavorite** – check if current symbol belongs to favorites (AFL 2.5)
- ◆ **IsIndex** – check if current symbol is an index (AFL 2.5)
- ◆ **Median** – calculate median (middle element) (AFL 2.5)
- ◆ **Percentile** – calculate percentile (AFL 2.5)
- ◆ **NumToStr** – convert number to string (AFL 2.5)
- ◆ **printf** – Print formatted output to the output window. (AFL 2.5)
- ◆ **StrFind** – find substring in a string (AFL 2.5)
- ◆ **StrFormat** – Write formatted output to the string (AFL 2.5)
- ◆ **StrToNum** – convert string to number (AFL 2.5)
- ◆ **SetFormulaName** – set the name of the formula (AFL 2.5)
- ◆ **LineArray** – generate trend-line array (AFL 2.5)
- ◆ **fclose** – close a file (AFL 2.5)
- ◆ **feof** – test for end-of-file (AFL 2.5)

- ◆ **fgets** – get a string from a file (AFL 2.5)
- ◆ **fopen** – open a file (AFL 2.5)
- ◆ **fputs** – write a string to a file (AFL 2.5)
- ◆ **RestorePriceArrays** – restore price arrays to original symbol (AFL 2.5)
- ◆ **SetForeign** – replace current price arrays with those of foreign security (AFL 2.5)
- **IMPROVED PLUGINS for:**
  - eSignal – improvements in handling international exchanges and daily mode*
  - QuotesPlus – now does not require 'administrator' account to be run on NT/2k/XP*

## Technical information

- [Troubleshooting guide](#)
- [Files used by AmiBroker](#)
- [Crash recovery system and automatic bug reporting](#)
- [Usage notes on new features](#)
- [Change Log](#)

### Troubleshooting guide

Quick jump: [AmiQuote](#), [data plugins](#) (eSignal, myTrack, IQFeed, QuoteTracker, Quotes Plus, TC2000)

#### CATEGORY: CRASH OR HANGUP

Where, where	Problem	Reason	Solution
<b>AmiBroker,</b> <b>Just after installation</b>	AmiBroker hangs at startup or frequent errors occur	Old/incompatible system components	<p>Download and install the latest FULL version of AmiBroker from <a href="http://www.amibroker.com/download.html">http://www.amibroker.com/download.html</a></p> <p>If you are running <b>Windows 95, 98, NT 4</b>: Install the latest version of the Microsoft run-time files and operating system components available <a href="#">here</a>.</p> <p>If you are running <b>Windows 95</b>: Please check if you have Internet Explorer 4 or higher installed (6.0 recommended)</p>
<b>AmiBroker,</b> <b>At startup</b> <b>(it was working fine before)</b>	AmiBroker crashes or hangs	Corruption of some data file	<p>Please try the following:</p> <ul style="list-style-type: none"> <li>• rename default database directory and try to run AmiBroker, if it works it means that some files inside this database are corrupted. You may send us DEFAULT.AWL file from "Layouts" subfolder for checking</li> <li>• delete or rename all DEFAULT.AWL files that you can find on your disk and try to run AmiBroker. (.AWL files are created inside "Layouts" subfolder of main AmiBroker database and "Layout" subfolders of each database directory</li> <li>• rename broker.charts and broker.bcharts files (in AmiBroker directory) and try to run AmiBroker</li> <li>• rename default.layout file (in AmiBroker directory) and try to run AmiBroker</li> </ul>
<b>AmiBroker,</b> <b>When running some AFL code</b>	AmiBroker crashes or hangs	Bug in AFL formula or in AmiBroker	Send offending code to AmiBroker support ( <a href="mailto:bugs@amibroker.com">bugs@amibroker.com</a> ) and delete / modify / comment-out it to continue to work until we find the reason of the problem and solution.



**CATEGORY: OTHER**

Where, when	Problem	Reason	Solution
<b>AmiBroker,</b> <b>Just after installation</b>	Help (User's guide) is not accessible	HTMLHelp system not installed	Install the update to the HTML Help system available from Microsoft <a href="#">here</a>
	Scripts do not work	Windows Scripting Host not installed	If you are running <b>Windows 95, NT 4.0</b> : Install Internet Explorer 5 to activate Windows Scripting Host, or install Microsoft Scripting engines available from <a href="http://msdn.microsoft.com/scripting/">http://msdn.microsoft.com/scripting/</a>
<b>AmiBroker,</b> <b>At startup (it was working fine before)</b>	Some feature does not work	Missing component of AmiBroker	Please check if you have not deleted any vital AmiBroker files. If you are unsure you may run the setup again over the old installation (please do NOT uninstall if you want to have your settings preserved)
<b>AmiQuote –</b> <b>Downloading quotes</b>	Download from Quote.com fails with 'the connection with the server is reset'	Livecharts password incorrect or not entered properly as described <a href="#">here</a>	Please go to the AmiQuote SETTINGS page, UNMARK "Use livecharts account" and select "SERVER #1" or follow exactly the instructions <a href="#">here</a>
	Download from Yahoo starts fine but then stops  (especially on DSL, ADSL, cable modem connections)	Yahoo is blocking you. After infamous Internet worm attack, Yahoo considers quick, repeating multiple downloads as a "denial of service" attack.	If possible reconnect with different IP (otherwise you would need to wait half an hour or more until Yahoo unblocks you)  go to the AmiQuote "Settings" and set "Initial delay between requests" to 1000 and "max number of simultaneous downloads" to 1.
	Download from Yahoo fails	It may be too soon to get historical data from Yahoo or  Yahoo temporary problem	First check Yahoo historical page:  <a href="http://table.finance.yahoo.com/k?s=utx&amp;g=d">http://table.finance.yahoo.com/k?s=utx&amp;g=d</a>  replace utx by the symbol in question. =====
			If the page shows old quotes – it is the problem with Yahoo not with AmiQuote.  In fact AmiQuote uses " Download spreadsheet format" link on previously mentioned page: <a href="http://table.finance.yahoo.com/k?s=utx&amp;g=d">http://table.finance.yahoo.com/k?s=utx&amp;g=d</a>

			<p>If historical data are not available you can always use "Current" mode of AMiQuote to get the data of today (even during trading hours)</p> <p>If you do not know the ticker symbol for index or stock or mutual fund please use symbol lookup feature:</p> <p>a) at Yahoo (for historical and current modes):  <a href="http://finance.yahoo.com/">http://finance.yahoo.com/</a></p> <p>b) at Lycos (for Intraday mode):  <a href="http://finance.lycos.com/home/misc/symbol_search.asp">http://finance.lycos.com/home/misc/symbol_search.asp</a></p>
	Download works but there are no quotes added	Local configuration problem	<p>Follow these steps to troubleshoot the problem</p> <ul style="list-style-type: none"> <li>• Go to C:\Program Files\AmiBroker\AmiQuote\Download and check if .AQH files are there.</li> <li>• Open them with Notepad and see if data are there</li> <li>• Check if you have aqh.format file in the Formats subdirectory of AMiBroker</li> <li>• Check its contents and eventually send me for checking</li> <li>• Try to import manually as per instructions given in the user's guide</li> <li>• If you ever restored data directory from CD ROM check the "read-only" flag of data files in AmiBroker database. They must not be read-only. If they are you have to mark all files in the data directory and all subfolders and UNMARK read only flag.</li> </ul>
	Download from any source fails	Connection problem	Please check your internet connection. If you are using firewall make sure to allow AmiQuote (QUOTE.EXE) to access the internet on ports 80 and 443.
<b>AmiBroker /eSignal plugin</b>	RT data spikes, bad ticks, other data problems.	Connection problem or other reason	Click with RIGHT mouse button over plug-in status area (green "OK" field) and choose "Fixup data for symbol" and wait a while. This will cause that entire intraday history for given symbol is re-downloaded.
<b>AmiBroker /myTrack /IQFeed /QuoteTracker plugin</b>	RT data spikes, bad ticks, other data problems.	Connection problem or other reason	Click with RIGHT mouse button over green "OK" field and choose "Force backfill" and wait a while. This will cause that entire intraday history for given symbol is re-downloaded.
<b>AmiBroker /QuoteTracker plugin</b>	Data does not update	Not enough 'ad-clicks' in QuoteTracker	<p>You have to click on advertisements in QuoteTracker or register it. For more details see:</p> <p><a href="http://www.amibroker.com/qthelp.html">http://www.amibroker.com/qthelp.html</a></p>

<b>AmiBroker</b> <b>/any RT plugin</b>	Disconnection	Connection problem	Click with RIGHT mouse button over plug-in status area (will show red "WAIT" or "SHUT") and choose "Disconnect" and then "Connect". If this does not help restart AmiBroker
<b>AmiBroker</b> <b>/QuotesPlus plugin</b>	Error message "Can not initialize C-TREE"	Quotes Plus database problem	Check if Quotes Plus own chart program works – if not it means that this is a problem in QP2 database module and you need to contact Quotes Plus technical support for help
<b>AmiBroker</b> <b>/TC2000</b> <b>/TCNet plugin, sometimes</b>	Error message "Variable or object not set"	Timing problem inside TC2000 own components.	The only partial workaround to this problem is to move all data from TC2000 CDROM to your hard disk (by Hard disk lists).

### Miscellaneous questions

Question	Answer
Must I use Internet Explorer? Can I use Netscape Navigator, Opera, (other browser) instead?	<b>You can use any browser.</b> In case of Windows 95 Internet Explorer 4 (or higher) installation is needed because it updates some of the Windows components (common controls library, HTML help, scripting) AmiBroker uses extensively. But after the installation you don't need to use Internet Explorer.
Where is the help file (manual)?	User's guide is available from Help->Help topics menu in AmiBroker (accessible also by F1 key)

## Files and directories used by AmiBroker

**AmiBroker main directory** – the directory where you installed AmiBroker. You can find main AmiBroker executable file (BROKER.EXE) there.

### PROGRAM FILES

- **Broker.exe** – main application file
- **CoolTool.dll, MiscTool.dll, Brokey.dll**– additional application files required by broker.exe
- **HTMLView.exe** – the report viewer, used to display backtest reports
- **emailer.exe** – a small add-on command line utility that sends user-defined alerts via e-mail
- **DBCAPI.DLL** – (present only if RT version is installed) – eSignal DBCAPI support library
- **Plugins** – the subfolder that contains [all plug-in DLLs](#).
- **unins000.exe and unins000.dat** – uninstaller program.
- **Broker.chm** – compiled HELP file

### DATA FILES

- **Data** – default database folder, contains **broker.master, broker.workspace**, 0–9, A–Z, '\_' **symbol data subfolders** and **Layouts** subfolder. [Read this](#) to learn more about AmiBroker database

concepts.

- **broker.master** – binary data file containing the table of all symbols available in the database. Used for quick loading of symbols. The table of symbols includes information about assignments of symbols to categories (markets, groups, sectors, industries, watch lists, indices and favorites). If you delete this file AmiBroker will re-create it because this information is available also in the individual symbol data files.
- **broker.workspace** – binary data file containing the information about the database settings (interval, data source used, etc), category names, global advance/decline data, etc. If you delete this file you will lost database settings and category names will be reset to defaults.
- **default.awl, \*.awl** – Amibroker Workspace layout files. Text files that contain the information about the [layouts and chart sheets](#). The default.awl file stored in "Layouts" subfolder inside database folder contains default LOCAL layout for this database. The default.awl file stored in "Layouts" subfolder of AmiBroker main directory contains default GLOBAL layout (used when there is no local layout present). If you delete this file then default layout will be generated based on default.layout file that is supplied with AmiBroker.
- **broker.charts** – binary file containing information about all custom indicators (including settings and formulas). If you delete this file you will loose all custom indicators.
- **broker.bcharts** – binary file that contains interpretation/additional code for built-in indicators. If you delete this file you will loose all interpretation code that is included with built-in indicators
- **broker.layers** – text file that contains information about [chart layers](#). If you delete this file layers will be reset to factory defaults.
- **broker.groups, broker.markets, broker.sectors, broker.industries** – text files that contain default names for groups, markets, sectors and industries. Used only at the database creation time. Later this information is stored per-database in broker.workspace file. If you delete them, AmiBroker will default to group *n*, market *n*, sector *n*, industry *n* names, where *n* is 1...256
- **broker.prefs** – binary file that contains user preference settings (available from [Tools->Preferences](#)). If you delete this file AmiBroker will reset to factory default settings
- **broker.params** – text file that contains persistent information about [user-defined indicator parameters](#).

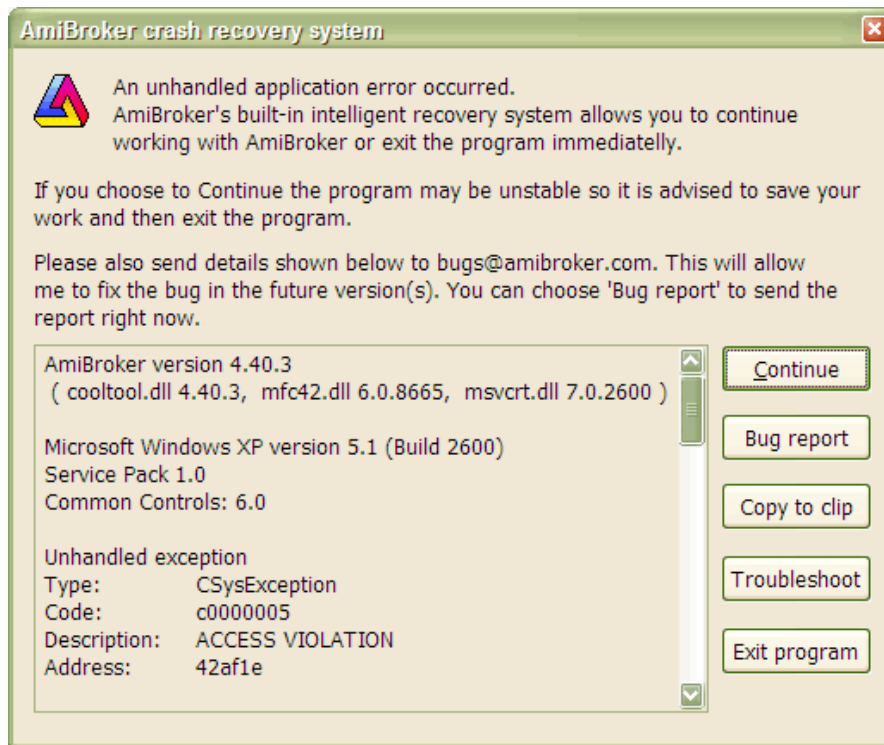
## Crash recovery system and automatic bug reporting

AmiBroker features a system of detecting and reporting bugs called "Crash recovery system". The name suggests that AmiBroker is now able to recover from such unexpected situations and indeed **it can!**.

How could this be done? Well... some tricks are needed to wrap the exception handling mechanism used by Windows :-)

Normally when Windows application performs some illegal memory access, illegal operation (for example division by zero) or illegal instruction the system pops the dead-end message box saying "This program has performed an illegal operation and will be shut down". Now you have got no choice – the application is terminated when you click on OK button.

AmiBroker's crash recovery system introduced in v3.47 beta intercepts the exception generated by Windows and instead of standard dead-end message box it displays the following dialog:



As you can see there is a window that displays important system information and there are three buttons **Continue**, **Bug report**, **Copy to clip**, **Troubleshoot**, **Exit program**. Clicking on the **Exit program** button works exactly the same as clicking on "OK" button in the standard Windows dead-end message box. But the first two buttons give you brand new possibilities. If you click on **Continue** button AmiBroker will try to recover from the error and continue running. In most cases you will be able to save your work and modifications you have made so you will not lose anything. In fact you will be able to work normally. There are however some cases when recovery will not succeed and AmiBroker may be unstable, so it is advised just to save your data and exit. It may also happen that this window will pop up for a couple of times – then you should just click on **Continue** several times.

The recover function is quite nice but the main purpose of this system is to find and fix the problems in future version and this is why the most important function was provided – **Bug report**. If you the crash recovery window popped up on your screen please click on **Bug report** button before attempting to continue work. This will automatically launch your default mail program and prepare the new e-mail with the following information:

```
To: bugs@amibroker.com
Subject: Bug Report (automatically generated by AmiBroker)

Steps needed to reproduce the bug:
>>PLEASE ENTER THE DESCRIPTION HERE<<

AmiBroker version 4.40.3
( cooltool.dll 4.40.3, mfc42.dll 6.0.8665, msvcrt.dll 7.0.2600 )

Microsoft Windows XP version 5.1 (Build 2600)
Service Pack 1.0
Common Controls: 6.0

Unhandled exception
```

Type: CSystemException  
 Code: c0000005  
 Description: ACCESS VIOLATION  
 Address: 42af1e

Line 1, Column 16:  
 ROC(Close,Null);  
 -----^

(occurred during AFL formula execution)

AFL Parser status:  
 Processing stage: EXECUTE  
 Formula ID: 0 ()  
 Action 3 (SCAN)

Additional information:

Number of stock loaded: 9331  
 Currently selected stock: WTEL  
 Number of quotes (current stock): 155

Workspace:  
 Data source = QP2 , Data local mode = 2, NumBars = 5000

Preferences:  
 Data source = (local), Data local mode = 1, NumBars = 1000

Command history:  
 2828 - Shows Analyser - a tool to test systems & explore  
 market--Auto-Analyser

Cache manager stats:  
 Number of list elements: 2  
 Number of map elements: 2  
 Hash table size: 5987

Memory status:  
 MemoryLoad: 52 %  
 TotalPhys: 523760K AvailPhys: 246560K  
 TotalPageFile: 1281044K AvailPageFile: 1050008K  
 TotalVirtual: 2097024K AvailVirtual: 1950604K

Last Windows message:  
 HWnd: 0x20eac  
 Msg: 0x0110  
 wParam: 0x00030ed6  
 lParam: 0x00000000

As you can see AmiBroker generates itself most important details for the bug report including even some history of menu selections (Command history) but the most essential thing at this point is to provide the description of steps needed to reproduce the bug. You should just type (in place marked by >>PLEASE

ENTER THE DESCRIPTION HERE<<) what you have done before the bug occurred, what special conditions must be met to reproduce it, maybe an AFL formula that you have tried and anything that you suppose might be important (even though AmiBroker includes a few lines of offending formula automatically). This is critical since automatically generated information is very nice but can not cover all the details. Then, when the bug report is complete just click **Send** in your mail program to send the report to me (note that address and subject fields are filled in for you).

Clicking **Copy to clip** button allows you to copy above report to the clipboard. You may use this button if automatic send does not work with your default e-mail program.

Clicking **Troubleshoot** brings up Troubleshooting page at <http://www.amibroker.com/troubleshoot.html> that contains descriptions of most common problems and how to solve them.

Some final notes: I have put significant amount of work in making this system reliable, however you should be aware that not all exception and/or system errors could be handled by this system and it may happen that AmiBroker will not be able to recover from some fatal error. It is also possible that this system would not be able to intercept all low level exceptions. In that case just prepare the report by yourself giving me as much details as possible.

Please remember that the final goal is making AmiBroker rock-solid and bug-free. This is what I am working on constantly.

## How to purchase AmiBroker ?

**AmiBroker is a trialware.** This means that you **SHOULD** evaluate the trial version of the program for a period of 30 days before buying it.

If you like the program and want to use it for more than 30 days evaluation period – you have to buy the license to use it. We assume that you installed AmiBroker before ordering and checked if it fits your needs.

**AmiBroker software is currently available in 2 editions: Standard and Professional (RT).** To learn about the differences between these two versions [click here](#).

### PRICING

One–time license fee is:

Edition	New Single–User License	Upgrade License (only for registered users of previous versions)
Standard	\$189 <b>Buy Now!</b>	\$89
Professional	\$259 <b>BuyNow!</b>	\$80 (from Standard Edition same version)
		\$119 (from earlier versions)

### BENEFITS:

Here is what **YOU** gain purchasing AmiBroker:

- **the keyfile** enabling all features of the program (database saving, no more annoying requesters)
- **free upgrades for one year from the date of purchase**
- access to **members–only zone** featuring
  - ♦ **AmiBroker Developer Kit** (for the developers of plugin DLLs)
  - ♦ newest issues **AmiBroker Tips weekly newsletter**
  - ♦ monthly **Stocks&Commodities® Traders' Tips for AmiBroker**
  - ♦ newest, private versions of AmiBroker
  - ♦ extra AFL formulas for indicators, commentaries, trading systems
- ability to influence the **future of AmiBroker** because your proposals of new features are much more likely to be implemented
- **50% discount** on next year of upgrade and maintenance pack
- 12 month **technical support** via e–mail
- other bonuses

### DELIVERY

After paying registration fee you will receive the **personalized keyfile by e–mail**. No other delivery methods are supported. **When purchasing please supply your e–mail address.**

### HOW TO ORDER AMIBROKER?

#### ORDERING ON–LINE

If you would like to buy AmiBroker, you can do the purchase online via ShareIt Secure Web Site (SSL) using links below. Payment methods include all major credit cards as well as cheques and wire transfers.



To place an order—on line, please visit: <http://www.amibroker.com/order.html>

AmiBroker online ordering is provided by SWREG.ORG (Atlantic Coast PLC) and ShareIt/element-5 AG a well established shareware registration and credit card processing agents. They handle registrations for over 7000 shareware programs. Their server uses your browser's powerful built in encryption and security, along with VeriSign/Thawte authentication, to encrypt your personal information and credit card details so that they cannot be intercepted by hackers or other third parties.

All credit card data are transmitted using the secure (encrypted) HTTP protocol according to the current SSL (Secure Socket Layer) 128-bit strong cryptography standard. We have all heard a lot of talk about whether shopping on the internet is safe. The fact is that this year on-line shoppers will spend over \$5.7 billion dollars according to International Data Corp. The main concern of on-line shoppers is that their credit card information will somehow end up in the wrong hands. SWREG.ORG and ShareIt/element-5 registration services use Secure Server technology, which encrypts your order information, keeping it private and protected. This technology is used by all the major commercial shopping sites. It is actually safer to transmit your credit card info over the Internet than it is to use your credit card around town.

For more information on security matters, please consult your browser's documentation. Also please note that all information submitted in the online shop is 100% confidential – we won't sell or give away your email address or other details!

#### **ADDITIONAL INFORMATION**

On-line purchasing is the fastest way to obtain your personal registration code(s). Once you complete your registration, you will receive your personal data within 24 hours.

It's of main importance that you give us a complete and correct Internet e-mail address. Entering an incorrect e-mail address (or an e-mail address that doesn't work correctly), you won't be able to register your software.

[E-mail us](#) if you have any further questions regarding registration, future versions, and so on.

# Credits

## **Thanks**

Many people make significant contributions to the development and testing of the AmiBroker. Thank you all. Thanks to Bill Barack, Rick Perkins, Ken Close, Dimitris Tsokakis, Marek Ch<sup>3</sup>opek, Herman van den Bergen, Ed Winters, Patrick Hargus, Mark Leon, Dan Clark, Rick Parsons, Charlie Hooper, Steve Wiser, Jim Ellis, David Holzgreffe, Carlton McEachern, Geoff Mulhall, Richard Cloonan, Peter Gialames, Stephane Carrasset, Dale Wingo, Fred Tonetti, Chuck Rademacher, Gary A. Serkhoshian, Rick Perkins, Tom Supera, Michael Robb, Mark Allen, Geo Singleman, Anthony Faragasso, Jayson Casavant, Al Holzwarth, Sidney Kaiser, William Peters, Ara Kaloustian and all the other AmiBroker users for giving me valuable feedback, comments, ideas, suggestions, test results and all the support. Special thanks to eSignal, myTrack, IQFeed, MarketCast for their generous support and co-operation. Thanks to Jerry Medved (QuoteTracker) for co-operation. Thanks to Mark Jurik of Jurik Research for providing his tools. Special thanks to Gary Lyben of Quotes Plus for the help with interfacing to Quotes Plus database. Thanks to Yuki Taga for proof-reading parts of tutorial section. Many thanks to all the contributors to the [AFL formula library](#) for sharing their work. Many thanks to [Sharenet](#) (Robin and Steve) and all South African users for their continuous support. Thanks to Jordan Russell and Martijn Laan for their InnoSetup/ISX. Thanks to Donald Dalley for extensive support he provided for Amiga version of AmiBroker. The deepest thanks and love to my wife Elizabeth and kisses for our little children Julia and Jacob for bringing so much joy to my life everyday. And thank You, Dear User, for purchasing AmiBroker! With your kind support we can make dreams come true.

## **AmiBroker on the Web**

For latest news, patches and updates please check out AmiBroker/Win32 WWW site at:  
<http://www.amibroker.com>. (The backup site <http://www.amibroker.net>)

Please check also AmiBroker Tips Newsletter available at: <http://www.amibroker.com/newsletter>

Visit support section of AmiBroker web page at: <http://www.amibroker.com/support.html>

Check AFL on-line library:

<http://www.amibroker.com/library/>

DevLog:

<http://www.amibroker.com/devlog/>

Knowledge Base:

<http://www.amibroker.com/kb/>

AFL on-line reference:

<http://www.amibroker.com/guide/afl/>

Third-part area (plugins, documentation):

<http://www.amibroker.net/3rdparty.php>

Check AmiBroker message boards at:

<http://www.amibroker.net/boards/>

Check AmiBroker mailing lists at:

<http://www.egroups.com/messages/amibroker-news> (announcements)

<http://www.egroups.com/messages/amibroker> (general discussion)

<http://www.egroups.com/messages/amibroker-ts> (trading systems)

<http://www.egroups.com/messages/amibroker-afl> (AFL coding)