Advanced users' workshop:

Custom Backtester Interface

by Tomasz Janeczko,
Amibroker.com

Custom backtester interface (CBI) - what for?

For everything that is not possible to do with standard backtester except....

making coffee

(feature not implemented, sorry)

Custom backtester interface (CBI) - what for?

- adding your custom metrics position sizing based on portfolio-level equity
- advanced scaling-in/-out based on portfolio equity (for example rebalancing) and other run-time stats
- customized rotational trading systems
- implementing custom formulas for slippage control
- advanced systems using PF-level stats on bar-by-bar basis to decide which trades to take

Purpose of this session

- to explain some basic concepts
- to show a couple of usage examples
- to answer questions that you may have
- it is **NOT** 1-hour programming course

Portolio backtest: 2 passes

- first pass
  - collecting trade signals,
  - ranking/sorting by position score

    (your AFL formula is executed **once for every symbol** under test)
- second pass
  - actual backtest (simulation of trading on historical data using signals collected in 1st pass)

(executed only once per backtest)

First backtester pass (regular)

RAW SIGNALS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
AAPL	Sell	Buy	Buy	Buy	Sell	Sell	Buy	Buy	Buy	Buy	Buy	Buy	Buy	Sell	Sell	Sell
MSFT	Sell	Sell	Buy	Sell	Sell	Buy	Sell	Sell	Sell	Sell	Sell	Buy	Sell	Sell	Sell	Buy
INTC	Buy	Buy	Buy	Buy	Buy	Buy	Buy	Buy	Buy	Buy	Buy	Sell	Sell	Sell	Sell	Sell
CSCO	Buy	Buy	Buy	Sell	Sell	Sell	Buy	Buy	Buy	Buy	Sell	Sell	Sell	Buy	Buy	Buy
LVLT	Sell	Buy	Buy	Buy	Buy	Sell	Sell	Sell	Sell	Buy	Buy	Buy	Sell	Sell	Sell	Sell
AMGN	Buy	Buy	Sell	Sell	Buy	Buy	Buy	Sell	Sell	Buy	Buy	Buy	Buy	Sell	Sell	Buy

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	16
AAPL		Buy(10)			Sell	Sell	Buy(8)							Sell	Sell	Sell
MSFT	Sell	Sell	Buy(6)	Sell	Sell	Buy(1)	Sell	Sell	Sell	Sell	Sell	Buy(1)	Sell	Sell	Sell	Buy(2)
INTC	Buy(3)											Sell	Sell	Sell	Sell	Sell
CSCO	Buy(5)			Sell	Sell	Sell	Buy(10)				Sell	Sell	Sell	Buy(1)		
LVLT		Buy(8)				Sell	Sell	Sell	Sell	Buy(3)			Sell	Sell		
AMGN	Buy(4)		Sell	Sell	Buy(3)			Sell	Sell	Buy(2)				Sell		Buy(3)

PHASE 2 - PICKING TOP TRADES - MAX OPEN POS = 2, TRADES PICKED HAVE HIGHEST SCORE, ONCE PICKED, REMAIN IN PLACE UNTIL SE

GREY COLOR MEANS SKIPPED TRADE

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
AAPL		Buy(10)			Sell	Sell	Buy(8)							Sell	Sell	Sell
MSFT	Sell	Sell	Buy(6)	Sell	Sell	Buy(1)	Sell	Sell	Sell	Sell	Sell	Buy(1)	Sell	Sell	Sell	Buy(2)
INTC	Buy(3)											Sell	Sell	Sell	Sell	Sell
CSCO	Buy(5)			Sell	Sell	Sell	Buy(10)				Sell	Sell	Sell	Buy(1)		
LVLT		Buy(8)				Sell	Sell	Sell	Sell	Buy(3)			Sell			
AMGN	Buy(4)		Sell	Sell	Buy(3)			Sell	Sell	Buy(2)				Sell		Buy(3)

Second backtester pass

This is where custom backtester interface can be used

- For each bar the following things happen:
  - Top ranked entry signals are checked and trades are entered (if funds are available)
  - Exit/scale signals are matched against open positions and executed
  - Stops are applied and executed
  - All portfolio-level statistics/metrics are updated

With CBI you can actually change every aspect of this pass

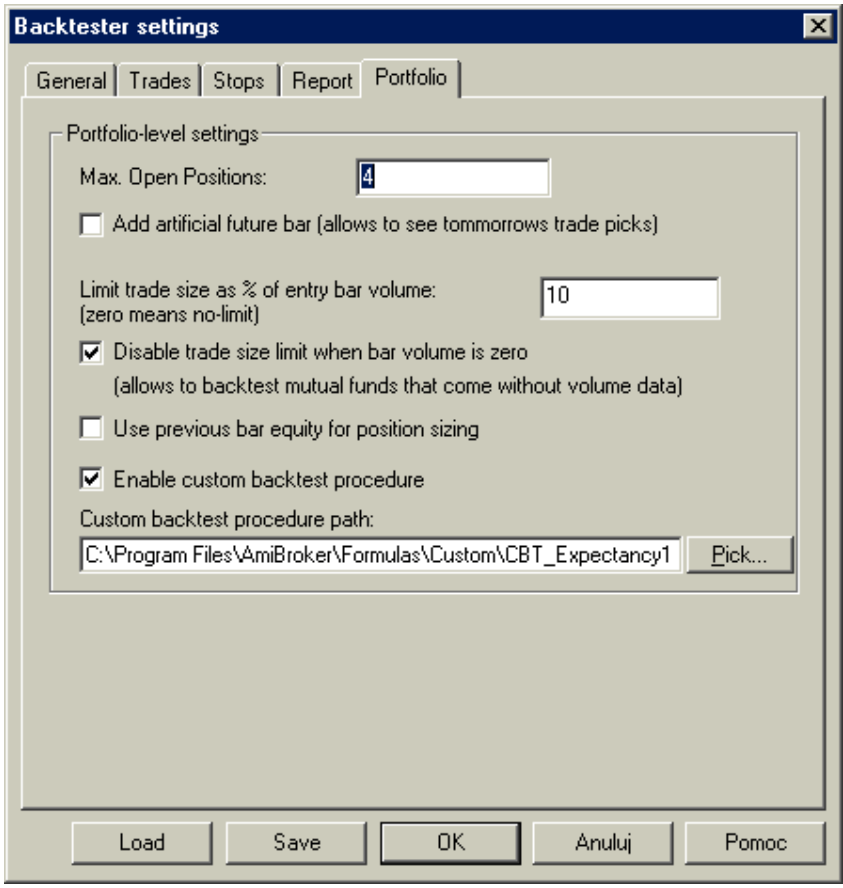
How to enable it ?

To enable custom backtest, you can use **AA->**

Settings,

Portfolio tab

(if you do so, custom code will be applied to ALL backtests)



How to enable it ?

...or you can enable it from the code:

```
SetOption("UseCustomBacktestProc", True );
```

or

```
SetCustomBacktestProc(
   "C:\\MyPath\\MyCustomBacktest.afl" );
```

(if you want to use use external file for it)

In this case custom backtest will be applied to current formula only.

Where to enter CBT code if it is enabled inside formula

To distinguish between normal run (phase 1) and final backtest run (phase 2) you need to use Status function:

```
SetCustomBacktestProc("");

if( Status("action") == actionPortfolio )
{
    ... YOUR CBT CODE (PHASE 2)HERE....
}

... YOUR REGULAR TRADING SYSTEM (PHASE 1)
HERE...
```

CBI - 3 programming levels

- **high-level - the easiest** (allows simple implementation of custom metrics)
- **medium-level** (allows to modify signals, query open positions - good for advanced position sizing)
- **low-level** approach (the most complex) - provides full control over entire backtest process for advanced programmers only

CBI programming model

- Custom backtester interface uses so called “object oriented programming” methodology (a.k.a. OOP)
- **Don’t be afraid** - at basic level (only this level is required to understand CBI) OOP is fairly simple

OOP - object definition

- In computer science an object is self-contained **entity** that encapsulates both data (so called properties) and procedures (so called methods) to manipulate the data.

- Sounds difficult.? .... Maybe but it is actually simple...

OOP - simple example

Before we dig into objects used by CBI one “real-world” example what object is and how to use:

- a PEN - in programming could be represented as object having
  - **properties**
    - color, thickness
  - **methods** that perform some actions
    - DrawLine(x, y) for example

```
pen = CreatePen(); // object creation
pen.thickness = 2; // property modification
pen.DrawLine( 30, 20 ); // method call
```

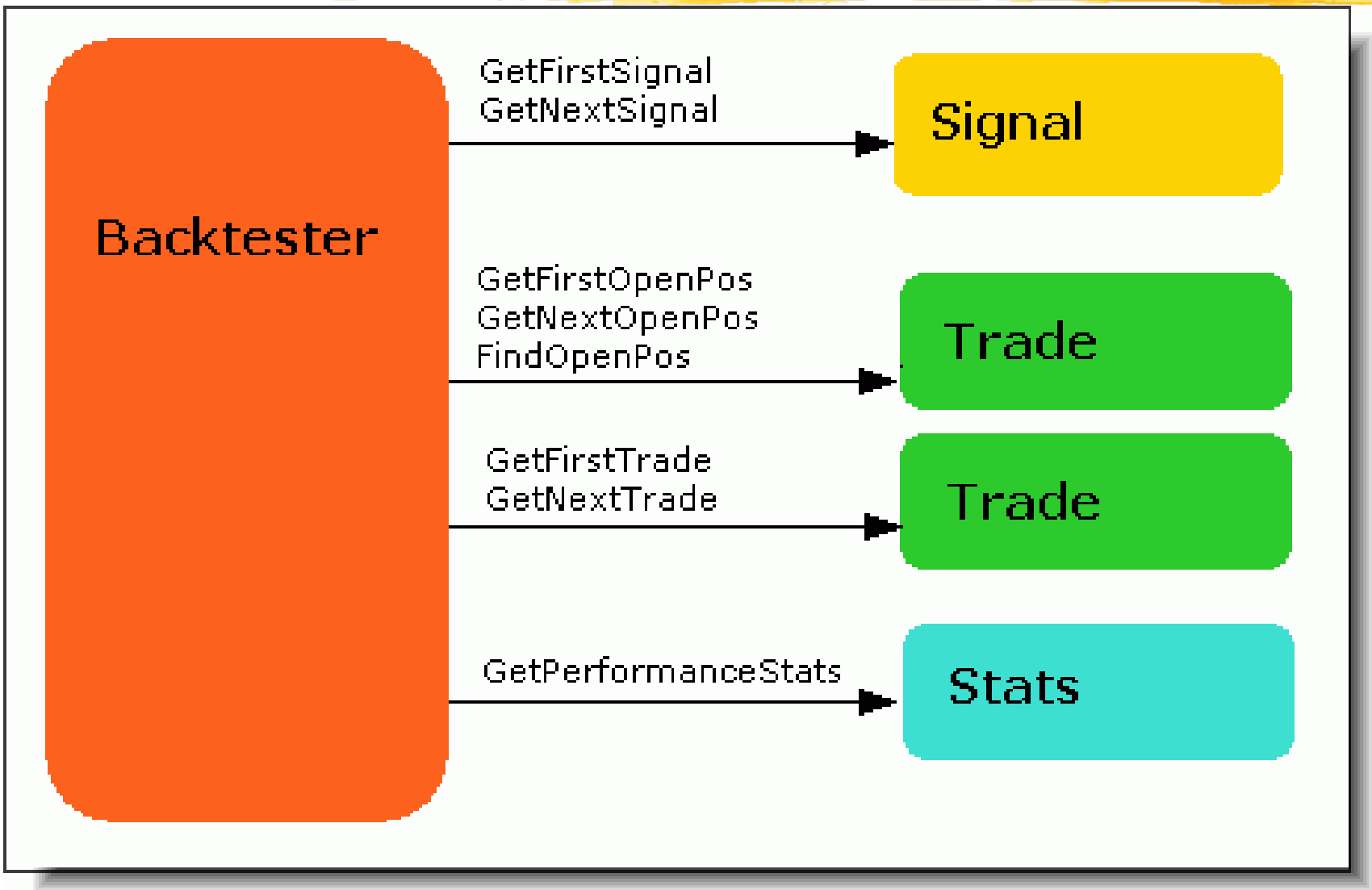
OOP vs functional programming

Many old-time programmers are afraid about OOP, while they used more or less the same idea without actually realising that.

Example:

FILE HANDLE -> OBJECT - in every programming language there is a concept of file handle that all file functions (**METHODS**) require to identify the file (**OBJECT**) on which to operate.

CBI object hierarchy



CBI access to objects

- Backtester object is available directly using GetBacktesterObject() AFL function.
- All other objects (Signal/Trade/Stats) are accessible by calling appropriate methods of backtester object

High level mode

- The simplest.
- Uses only two objects (**Backtester** and **Stats**) and only two methods (**Backtest**()/**GetPerformanceStats**())
- how does it work?
  - We call default **Backtest()** procedure
  - and after that we are collecting statistics to calculate our own figures.
- what for?
  - user-defined portfolio-level metrics

Ex 1: High Level - custom metrics

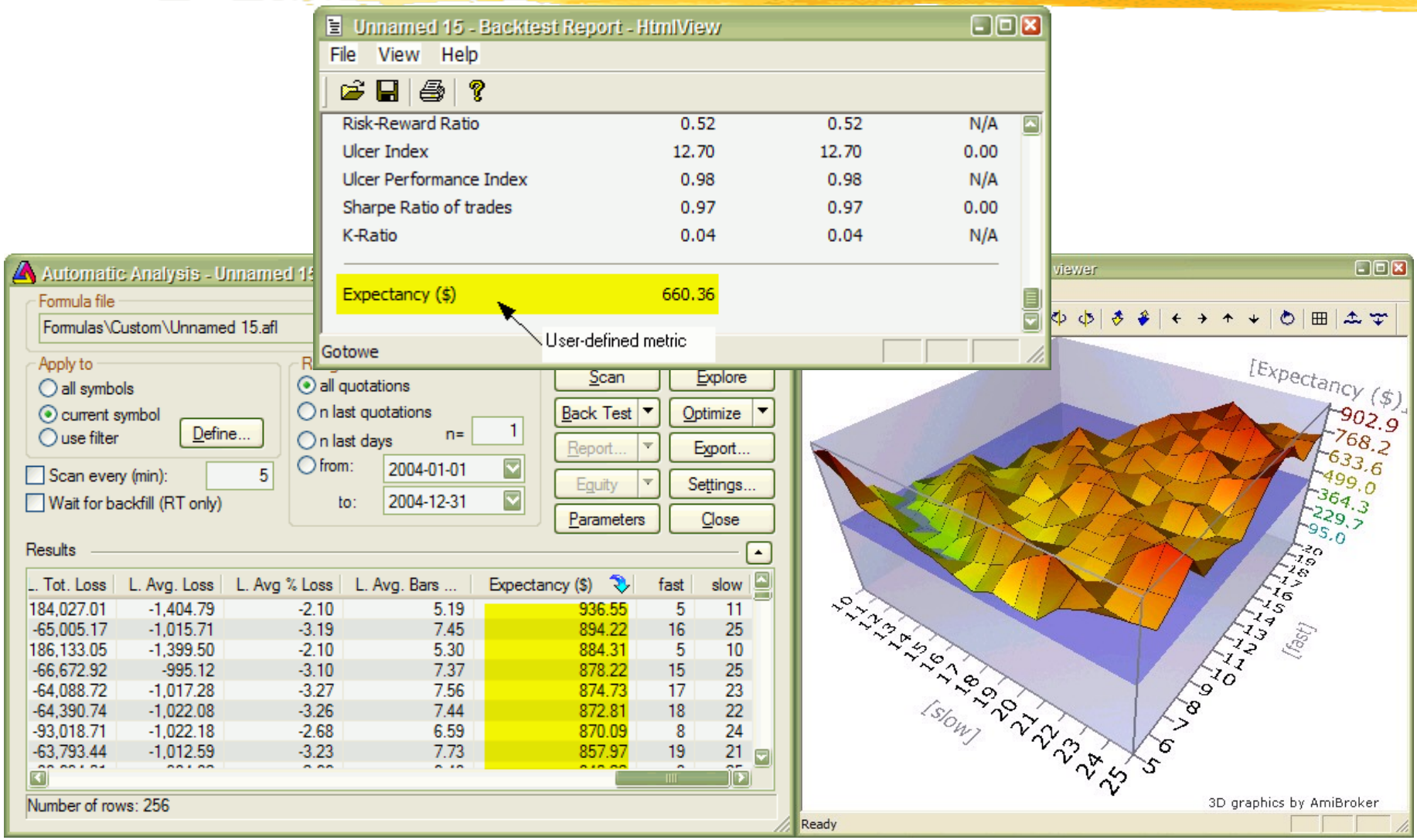
- In the first example we will add simple new metric to backtest/optimization output:

  Expectancy ($) = %Winners * AvgProfit - %Losers * AvgLoss

Ex 1: High level - custom metrics - cont.

```
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 =
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 );
}
```

Ex 1: High level - custom metrics - results



Medium level

- Semi-advanced - uses all object classes
- how does it work?
  - for each bar:
    - we can modify signals, check/modify open positions, retrieve per-trade statistics
    - then we call default signal processing method
- what for?
  - Advanced position sizing
  - PF-level signal control (custom rotational trading)
  - Trade-based metrics

Ex 2: Mid-level - pos. sizing based on portfolio eq.

```
if( Status("action") == actionPortfolio )
{
   bo = GetBacktesterObject();
   bo.PreProcess();
   for( bar = 0; bar < BarCount; bar++ )
   {
     CurrentPortfolioEquity = bo.Equity;
     for( sig = bo.GetFirstSignal( bar ); sig; sig = bo.GetNextSignal( bar ) )
     {
       if( CurrentPortfolioEquity > 50000 ) sig.PosSize = -20;
       if( CurrentPortfolioEquity > 60000 ) sig.PosSize = -16;
       if( CurrentPortfolioEquity > 80000 ) sig.PosSize = -12;
     }
     bo.ProcessTradeSignals( bar );
   }

   bo.PostProcess();
}
```

Ex 3: Mid-level - excl. top-N signals in rotational mode

```
SetOption("UseCustomBacktestProc", True );
ExcludeTopN = 1; // how many top positions to exclude
if( Status("action") == actionPortfolio )
{
    bo = GetBacktesterObject();
    bo.PreProcess();
    for( bar = 0; bar < BarCount; bar++ )
    {
       Cnt = 0;
       for( sig = bo.GetFirstSignal( bar ); sig; sig = bo.GetNextSignal( bar ) )
       {
         if( Cnt < ExcludeTopN ) sig.Price = -1; // exclude
         Cnt++;
       }
       bo.ProcessTradeSignals( bar );
     }
     bo.PostProcess();
}
EnableRotationalTrading( True );
SetOption("MaxOpenPositions", 5 ); SetOption("WorstRankHeld", 10 );
PositionSize = -20; PositionScore= 1/RSI(14);
```

Low level mode

- The most complex but most powerful
- how does it work?
  - for each bar
    - we can check signals/open pos/PF-stats to decide what trades to enter/exit/scale
    - we can call EnterTrade/ExitTrade/ScaleTrade for using any parameters we want, we are not limited by signals
    - we need to handle stops and update portfolio statistics
- what for?
  - rarely used, only for very advanced pf systems

Ex 4: Mid/Low-level - rebalancing

```
if( Status("action") == actionPortfolio )
{
   bo = GetBacktesterObject();
   bo.PreProcess(); // Initialize backtester
   for(bar=0; bar<BarCount; bar++)
   {
      bo.ProcessTradeSignals( bar );       CurEquity = bo.Equity;
      for( pos = bo.GetFirstOpenPos(); pos; pos = bo.GetNextOpenPos() )
      {
         posval = pos.GetPositionValue();
         diff = posval - 0.05 * CurEquity; // rebalance to 5% of pf equity
         price = pos.GetPrice( bar, "O" );
         if( diff != 0 AND abs( diff ) > 0.005 * CurEquity
                       AND abs( diff ) > price )
         {
           bo.ScaleTrade( bar, pos.Symbol, diff < 0, price, abs( diff ) );
         }
      }
   }
   bo.PostProcess(); // Finalize backtester
}
```

Some questions I collected before (1)

Q: Rebalancing sample: can the weight also be an array, so the weights become dynamic?

A: Yes it can. Instead of this line:

```
diff = posval - 0.05 * CurEquity;
```

Use this:

```
diff = posval - Foreign("~TickerWithWeights", "C") *
   CurEquity;
```

Some questions I collected before (2)

Q: How can I access percentage position size to make leverage adjustment for expectancy per $100 invested

A: You need to store original percent position size from appropriate Signal object (if you are using regular mode). To do so, you can use SetVar function inside loop using mid-level

```
for( sig = bo.GetFirstSignal( bar );
   sig;
   sig = bo.GetNextSignal( bar ) )
   VarSet("OrigSize" + sig.Symbol, sig.PosSize );
```

Later on you would need to read it back when you iterate through trades.
Because of complexity I will post code sample a bit later to the KB.

Some questions I collected before (3)

Q: I have problem with using ATC in CB procedure together with atcFlagEnableInPortfolio

A: Yes there is a problem in current beta, but it will be fixed next week

Some questions I collected before (4)

- Q: Is there already a way to automatically save the "~~~EQUITY" to a different choosen name after a backtest? If not, would you consider introducing this possibility?
- A: Currently there are two ways:
  - harder: writing equity to file and using OLE to re-import it at the end of CB procedure.
  - easier: using ATC and atcFlagEnableInPortfolio (but as mentioned earlier it requires fix to the beta)

Some questions I collected before (5)

- Q: Will there be a link between the account manager and the portfolio-BT/CBT
- A: At some point in the future yes. First version of new account manager that will appear within months probably will not have it.

Any more questions?

- Please feel free to ask any questions...

Thank You

For more information visit:

http://www.amibroker.com