TradeAngle Language

Motivation – TradeStation EasyLanguage method of trading strategies is slow, and cumbersome (for the programmer). TradeStation wants users to trade through its platform so they have crafted their scripting language to have major gaps in its functionality and information capabilities. There is no ability, for instance, to know specific information about a given open order. The programmer has to figure out which if his (possibly many) trades have fired within the last ‘bar’ by looking at OHLC values. Stop and Limit orders are cancelled *on the next bar*. Again the programmer has to figure out if the stop/limit order has actually been hit, and if not then re-fire the *same* order again for every bar.

It is also very tightly geared to their graphical environment. The optimization of any given trading strategy is brute-force (ie O(n^3) for 3 variables), and is unusable for anything but the simplest of algorithms. And finally, there is no debugging capabilities built-in to the EasyLanguage compiler. Error messages like ‘an array bounds has been breached’ without specifying a line number or variable name are useless and add many hours of debugging to complicated scripts.

Result – TradeAngle Language (TAL) has been created to take the trading strategy outside of TradeStation. We have created the following:

TAL Specification : the script language is specified using a BNF Grammar. It is comparable to Pascal with a little C

TAL Parser/Compiler : the script is parsed and compiled into a binary file. Any errors are detected and displayed showing their precise location within the script

TAL System Functions : a library if internal ‘functions’ that can be called from within scripts is continually being updated. Useful functions range from simple square-root calculations, to more complex stochastics

TAL Simulator : the compiled script can be ‘run.’ Currently we have (ironically) tied the running of a TAL script back into TradeStation. This allows us the graphical interface for results, strategy monitoring capabilities, and the price data.

The next stage (in development now) is a stand-alone simulator that does not require TradeStation to run TAL files. It will include:

(1) Connectivity to a quote stream (InstaQuote, HyperFeed, etc.) for price data
(2) Smart optimization of script variables (Monte Carlo method, etc)
(3) More trade types (Trailing Stop, Banded, etc) with *script access to trades*
(4) Portfolio handling (with library functions to allow TAL scripts to access portfolio/account information)
(5) Back-testing capabilities
(6) Graphical interface

Here is a simple TAL script example:

```
--------------------------------------------------------------------------
Inputs:
        D1 : DataSeries;
        ProfitPoints : double;
        LossPoints : double;

Var: ma : double;
Var: p1,p2 : double;
Var: Trades : intarray;
Var: i,t integer;

ma = MovingAverage(D1.Close(),30);

if ma > D1.Open[b1] then begin
        p1 = D1.CurrentPrice();
        i = NewTrade(GetUser(),D1.symbol(),p1,”B”);
        if i >= 0 Then begin
                Trades.add(i);
                SetLimit(i,ProfitPoints);
                SetStop(i,LossPoints);
        end;
end;

for i = 0 to Trades.Count() – 1
Begin
        t = Trades[i]l
        if(TradeStatus(t)== False) then
                Trades.remove(t);
        else
                Output(“Trade: “ + TradeName(t) + “ is live. P/L = “ + TradePL(t));
End;
```
