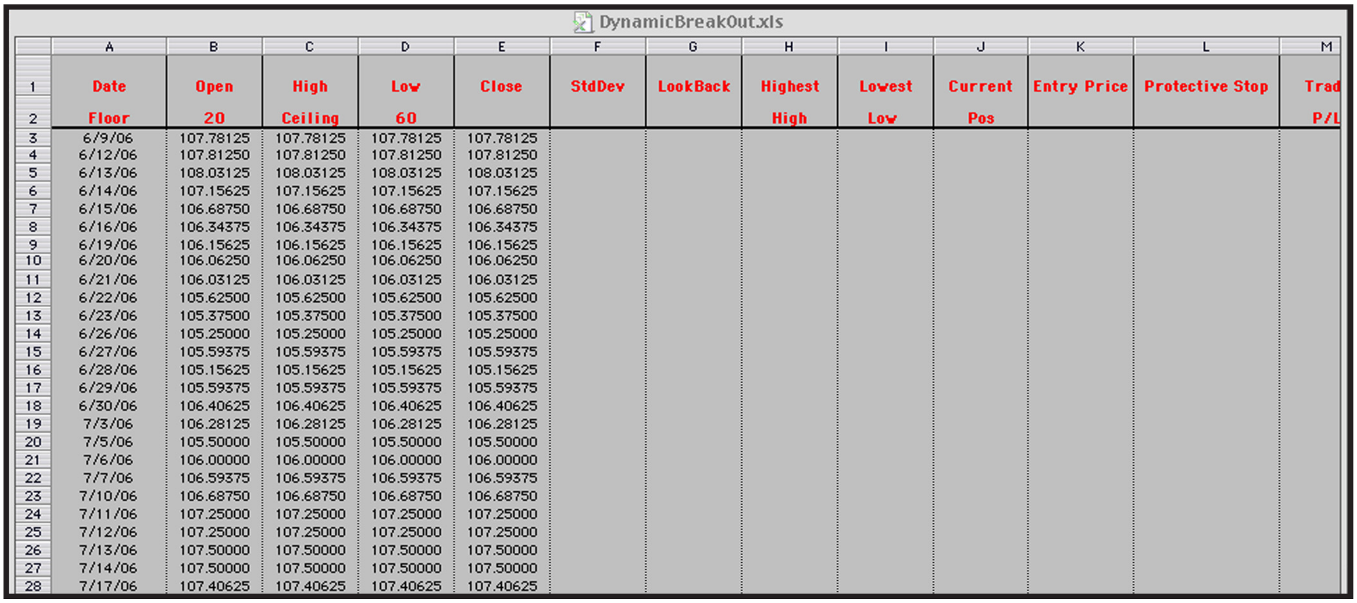
Using Excel's Visual Basic to Test & Trade Your System - Part 1

Back testing and trading a system real time is easy if you have access to TradeStation™, TradersStudio, Ninja Trader or any of the other available commercial applications. If you don't then its pretty much up to you and a calculator or spreadsheet. In a lot of cases this is all you need to calculate tomorrow's orders. However, back testing using these simple tools can be a tremendous nightmare and lead to inaccurate results. For the past 18 years I have worked on creating this type of software and I thought it would be fun to dedicate the next three installments of George's Corners to the development of a trading system order generator and back tester using tools that are generally found on any computer: Excel and Visual Basic.

Microsoft Visual Basic for applications is found in Microsoft Word and Excel and be found under the tools menu of these applications. You can also buy the Visual Basic IDE (Integrated Development Environment) that can build standalone applications. For our purposes we don't need that type of power and we want to use what we already have. If you have Microsoft Excel, then you are in business. Well almost. Some knowledge of programming is necessary to get the full benefits of the next three installments. If you are just reading this to get yourself acquainted with the concepts, then you will also be rewarded by receiving the Dynamic Break Out System (DBS) Excel workbook.

We will tip toe into the building of our software by first writing the code for generating the next day's orders for the DBS. Before we can write one line of code we need to set up a spreadsheet with data and some column headings. I have done most of the tedious work for you. Go to ftp://www.futurestruth.com/pub and download the DynamicBreakOut.xls file to your computer and open it up. It should look something like this. I simply created some headings for the information that I thought would be needed to help trade the system and imported the September 2007 contract of the bonds. I included the date, open, high, low and close data in the spreadsheet. Before I can explain the other column headings you will need to understand how the DBS system works.



DynamicBreakOut.xls

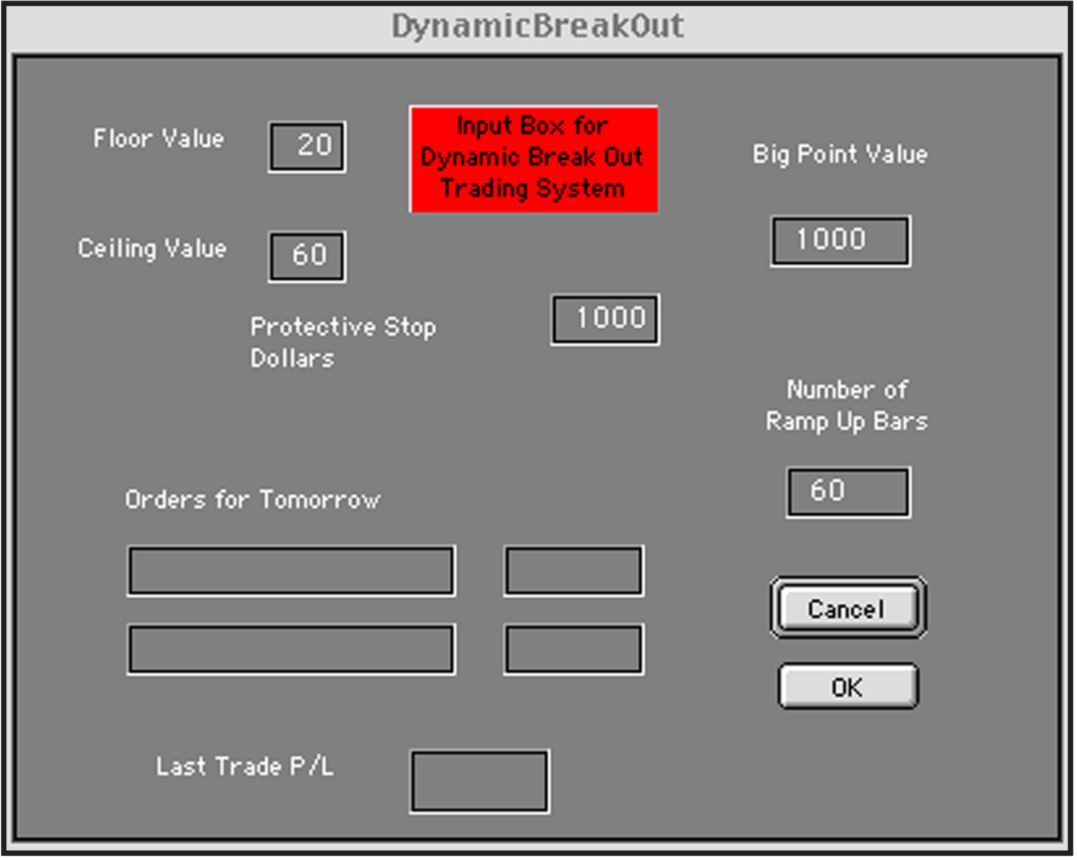
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Date	Open	High	Low	Close	StdDev	LookBack	Highest	Lowest	Current	Entry Price	Protective Stop	Trad
2	Floor	20	Ceiling	60				High	Low	Pos			P/L
3	6/9/06	107.78125	107.78125	107.78125	107.78125								
4	6/12/06	107.81250	107.81250	107.81250	107.81250								
5	6/13/06	108.03125	108.03125	108.03125	108.03125								
6	6/14/06	107.15625	107.15625	107.15625	107.15625								
7	6/15/06	106.68750	106.68750	106.68750	106.68750								
8	6/16/06	106.34375	106.34375	106.34375	106.34375								
9	6/19/06	106.15625	106.15625	106.15625	106.15625								
10	6/20/06	106.06250	106.06250	106.06250	106.06250								
11	6/21/06	106.03125	106.03125	106.03125	106.03125								
12	6/22/06	105.62500	105.62500	105.62500	105.62500								
13	6/23/06	105.37500	105.37500	105.37500	105.37500								
14	6/26/06	105.25000	105.25000	105.25000	105.25000								
15	6/27/06	105.59375	105.59375	105.59375	105.59375								
16	6/28/06	105.15625	105.15625	105.15625	105.15625								
17	6/29/06	105.59375	105.59375	105.59375	105.59375								
18	6/30/06	106.40625	106.40625	106.40625	106.40625								
19	7/3/06	106.28125	106.28125	106.28125	106.28125								
20	7/5/06	105.50000	105.50000	105.50000	105.50000								
21	7/6/06	106.00000	106.00000	106.00000	106.00000								
22	7/7/06	106.59375	106.59375	106.59375	106.59375								
23	7/10/06	106.68750	106.68750	106.68750	106.68750								
24	7/11/06	107.25000	107.25000	107.25000	107.25000								
25	7/12/06	107.25000	107.25000	107.25000	107.25000								
26	7/13/06	107.50000	107.50000	107.50000	107.50000								
27	7/14/06	107.50000	107.50000	107.50000	107.50000								
28	7/17/06	107.40625	107.40625	107.40625	107.40625								

Using Excel's Visual Basic to Test & Trade Your System - Part 1

I developed the DBS back in 1996 for a series of articles in *Futures* magazine and the system has done fairly well since. The backbone of the system is the old Donchian N week breakout; buy the highest high N weeks back and sell short the lowest low N weeks back. Instead of using a fixed N value or an optimized N value for the different markets, I decided to let the market tell the system the value of N. This was accomplished by the use of an adaptive algorithm; a method that changes itself based on available resources. My algorithm was based on the notion that market noise was directly related to volatility; the more volatility the more noise. Systems seem to get confused during noisy markets so I increased N when the markets exhibited higher levels of volatility and decreased N when the markets quieted down. As N increased so did the look back period which in turn pushed the buy and sell points further away from the current market. A decrease in N brought the buy/sell points closer and this occurred during periods of low volatility. A fixed money management stop was also included in the system even though it didn't help the overall results that much.

Now let's look back at the column headings in the spreadsheet. Some of the columns are self explanatory so I won't go over them. **StdDev** shows the standard deviation for the past thirty days. **LookBack** shows the value of N - as you can see it moves in the same direction as the standard deviation. **Highest High** and **Lowest Low** columns show the highest high and lowest low for past N (Look Back) days. **Current Pos** shows the current position of the DBS system. **Entry Price** shows the current trade's entry price. **Protective Stop** gives the current money management stop price and **Trade P/L** shows the profit or loss of the closed trades. I also have a **Floor** and **Ceiling** cell designation. Because we are adjusting N (lookback value) based on the adaptive algorithm, we need to put constraints on this value. We don't want to look back further than 60 days nor less than 20. I did not need to include all these columns in this spreadsheet to calculate the next day's trade signal, but I thought it would help with debugging and trading the system. We can simply look at any date and see what the standard deviation, highest high, lowest low, current position and protective values were.

Before we jump into the code let's go ahead and run the DBS macro (VB's nomenclature for sub program) so we can see the end result. Make sure you have the DynamicBreakOut workbook open and go under the tools menu. From there go to Macro and slide over to the sub menu and select Macros. This should bring up the Macro dialog box with RunDBS listed at the top. Click the Run button and the Dynamic BreakOut dialog box will come up and



Floor Value 20
Ceiling Value 60
Protective Stop Dollars 1000
Big Point Value 1000
Number of Ramp Up Bars 60
Last Trade P/L

Using Excel's Visual Basic to Test & Trade Your System - Part 1

look something like this. Here you can change the **Floor, Ceiling, Protective Stop, Big Point Value** and the **Number of Ramp Up Bars**. The Floor and Ceiling values limit the N-day look back value. In the default case the value can't go below 20 nor above 60. The protective stop is set to $1000 and since we are working with the US bonds the Big Point Value is also set to $1000. The Big Point Value is the dollar amount of a 1.00 move. So a move from 106.01 to 107.01 would be equivalent to a $1000 move. The **Number of Ramp Up Bars** is the value of the number of days necessary to calculate the trade signals. Since the N-day value can be 60 days, we set this value to 60. The other five text boxes are where the information for the next day's trade signal is presented. In this case we will be liquidating our short position at 111.6875 on a stop. Since we are working with the bond market, you would need to convert this to 32nds before you placed the stop order. We currently have $3656.25 in OTE. If you wanted to use this spreadsheet to generate your next day's trading signal you would need to add another line of data in the spreadsheet and then re-run the macro. This software isn't that sophisticated yet, but it is sufficient enough to help you with generating your signals without error. The best part is it's free, and when we are done, adaptable to any end of day trading system.

Let's go ahead and ease into some of the Visual Basic code. The first few lines of code are preceded by a single quote ('). This lets the VB interpreter know the line or lines following are comments and not to be translated into machine language. These comments make the code easier to read and understand for someone else other than the programmer. So the first four lines are just their to explain what the program is doing and who wrote it and when they wrote it.

```
'Subroutine to calculate the Buy and Sell Signal for the
'Dymanic Break Out System
'Template can be used to program any system
'programmed by George Pruitt in June 2007
```

The next line gives the subroutine a name and the names of the variables that will pass back and forth from the Dynamic Break Out dialog box. All of the information that is included in the dialog box is communicated to the subroutine through this one line of code.

```
Sub DBSSub(flr, clg, orderString1, orderString2, orderPrice1, orderPrice2, lastPL, protStop, big-
PointValue, rampUP)
```

```
'Name of the subroutine <sub program> that will calculate the signals.
'We will pass <send and receive> these arguments from/to the dialog box.
'   flr - the minimum look back period
'   clg - the maximim look back period
'   orderString - in english what we will do
'   orderPrice - the price where we will take action
'   lastPL - current trade profit or loss
'   protStop - what stop in points is being used
'   bigPointValue -$ value of a full point move-$1000 for bonds-$1250 for currencies
'   rampUp - how many days needed for calculations before a trade can take place
```

The next lines of code starting with the keyword DIM is where we declare the variables that we will be using in the subroutine. The names are somewhat explanatory - myDate(500) is declaring a list of 500 items that will hold the dates of all the data. Dim is simply tell VB to reserve enough space to handle 500 dates,

Using Excel's Visual Basic to Test & Trade Your System - Part 1

opens, highs, lows, closes and any other variables that we may need. If we have more than 500 rows of data, then we would need to change the 500 to whatever we may need plus another 100 or so. It's better to reserve more space than we need, because if we don't it can lead to major computer errors.

```
Dim stdArr(29), myDate(500), myHigh(500), myLow(500), myOpen(500), myClose(500) As Double
Dim cnt, i, j, length, hh, ll, dataCnt, baseRow as Integer
Dim pos, price, trdProfit, stoppedOut As Integer
Dim stopPrice, myPrice, myPos, longLiqPrice, shortLiqPrice as Integer
Dim buyStop, sellStop, displayRow As Integer
Dim delta, realLength, stdDev, prevStdDev As Double
```

Because we have two rows reserved for column headings we need to skip these two rows and we need to select the first sheet in the workbook.

```
baseRow = 3               'Where the actual data starts
dataCnt = 0
Sheets("Sheet1").Select     'Select Sheet1 to access the data
```

The next lines are key to understanding how VB works with cells and the lists we have created to hold the data. A cell, as we all know, is referred to by first is row and then its column. The first cell in any spreadsheet is located at row 1, column 1. This can also be written as Cells(1,1). Cells is a keyword built into VB and the parentheses are necessary. The first number inside the parentheses is the row location and the second is the column location. Again, I have made several comments inside the code to help explain what we are doing here (see the (') at the beginning of each line).

```
'Read in the data
'Column 1 - Date  -- Cells(3,1) - Cells(Row,Column)
'Column 2 - Open  -- Cells(3,2)
'Column 3 - High  -- Cells(3,3)
'Column 4 - Low   -- Cells(3,4)
'Column 5 - Close -- Cells(3,5)
```

Column 1 holds all of the different dates and column 2 holds all the opening prices, etc.,. See how the first date is referenced by Cells(3,1). Here we are saying the first date is located in row 3 column 1 (3,1). The first open price is located in row 3 column 2 (3,2). To get the data into our arrays (lists) we must loop through every row that contains valid data. Since we won't know exactly how many times to loop, we can loop until we find a row without any data. This is exactly what we are doing with our Do While loop. Notice how dataCnt starts at zero - VB allows zero based arrays, which simply means the first element (item) in our list can be indexed (referred to) by the number 0. The first date in the spreadsheet is in Cells(3,1), whereas the first item in our myDate list is myDate(0).

```
Do While Cells(dataCnt + baseRow, 1) <> ""
    myDate(dataCnt) = Cells(dataCnt + baseRow, 1)
    myOpen(dataCnt) = Cells(dataCnt + baseRow, 2)
    myHigh(dataCnt) = Cells(dataCnt + baseRow, 3)
    myLow(dataCnt) = Cells(dataCnt + baseRow, 4)
    myClose(dataCnt) = Cells(dataCnt + baseRow, 5)
```

Using Excel’s Visual Basic to Test & Trade Your System - Part 1

```
   dataCnt = dataCnt + 1
Loop
```

Here we start collecting the information that was provided in the dialog box - floor, ceiling, protective stop, rampUp, and Big Point Value. We also put the flr and clg values into the Floor and Ceiling cells on the spreadsheet. We initially set length (the number of days that we look back to obtain the highest/lowest high/low values) to the flr value (20), and we set our pos to 0 (flat), our buy stop price to 999999 and our sell short stop to 0.

```
Cells(2, 2) = flr 'Plug in the floor input into this cell on the spreadsheet -- for display
Cells(2, 4) = clg 'Plug in the ceiling input into this cell on the spreadsheet

length = flr
pos = 0              'Start us out flat


protStop = protStop / bigPointValue

longLiqPrice = 999999
shortLiqPrice = 0
buyStop = 999999
sellStop = 0
stdDev = 1
```

Now that we have read all of our data into our arrays we need to loop through each day and calculate our buy/sell points and see if a trade has occurred. We need to skip past the number of ramp up days so that we can lookback in time and calculate our standard deviations, highest highs and lowest lows. We will also start inputting our calculations into cells on the spreadsheet. Since our lists are not exactly in synch with our cells index, we must off set our cells by the baseRow value.

```
displayRow = rampUP + baseRow    'Start on the first date after the ramp up period
i = rampUP                       'Array index starting at the first day after the rampUp
Do While Cells(displayRow + 1, 1) <> "" 'Do while there is something in the first column
```

So we will index our data arrays with i and our cells with displayRow. The next step is to calculate the standard deviation for the past thirty days. This is done in the “for-next loop” below. We know how many days (iterations) we are looking back so we can use a “for loop”. So we start at our current i value and go back into time 30 days and load a temporary list with 30 elements of closing prices.

```
'      start looping through all data
'      Setup calculations
       cnt = 0
       For j = i - 29 To i
           stdArr(cnt) = myClose(j)  'load the last 30 days of closing prices into the
cnt = cnt + 1      'stdArr Array or List
       Next j
```

Using Excel's Visual Basic to Test & Trade Your System - Part 1

After the loop we pass our list to the StDev worksheet function and it passes back the standard deviation of the data sample.

```
prevStdDev = stdDev
stdDev = Application.WorksheetFunction.StDev(stdArr)'pass the list to the Standard
```

We then store the standard deviation in the column #6. Notice how the variable displayRow is used to reference the row and a fixed number is used for the column. The column number stays constant whereas the row number increases as we go through the different days of data. The next few lines of code is the adaptive algorithm. Notice how I determine the change in standard deviation from day to day. I take today's standard deviation and subtract the previous SD (standard deviation) and then divide by today's SD. This results in either a positive or negative number. I then add 1 to this value and then multiple by the previous days length. If the SD goes up then the length goes up - if the SD goes down then the length goes down.

```
Cells(displayRow, 6) = stdDev
delta = 1
delta = (stdDev - prevStdDev) / stdDev + 1  'calculate the delta of the Std. Devs
length = length * delta                    'change the lookback length
length = Application.WorksheetFunction.Round(length, 0)
                                           'we are working with whole numbers for lookback
Cells(displayRow, 7) = length                'put the length variable in column 7
If (length < flr) Then length = flr           'make sure our length is not too short
If (length > clg) Then length = clg           'make sure our length is not too long
```

Now that we know the value of the look back or N, we can set up our loop to look back in time to attain the highest high and lowest low values that will become our buy and sell stops. Again we know ahead of time how many days or iterations that we will need to look back, so we can use a for-next loop.

```
hh = 0
ll = 999999
For k = i - (length - 1) To i'start at the current day and look back lenght period
    If (myHigh(k) > hh) Then hh = myHigh(k)   'get the highest high length lookback
    If (myLow(k) < ll) Then ll = myLow(k)     'get the lowest low length lookback
Next k
Cells(displayRow, 8) = hh          'put the highest high on the sheet in column 8
Cells(displayRow, 9) = ll          'put the lowest low on the sheet in column 9
```

Once we loop back N-days we can store the highest/lowest high/low values in column 8 and 9 respectively. If you are unaccustomed on how these loops work, just send me an email and I will give you a quick tutorial. The next lines of code is really the meat of the order generation and soon to be back testing software.

Lets see if a trade has occurred! Each iteration of a loop signifies a day of data, so we start at the end of the day with a new buy and sell stop. We also set a temporary flag (stoppedOut) to determine if we get stopped out of a trade.

```
stoppedOut = 0
buyStop = hh
```

```
    sellStop = ll
```

First off we check to see if we our protective stop is closer to the market than our reversal stop. Here is what happens if have a long position: check to see if the next bars (i is the current bar index so i + 1 would be the next bars index) low price is below or equal to our protective stop and our protective stop is closer than our reversal sell stop. If it is then we change our pos variable to 0 (flat) and set our price to our protective stop price. We do need to check to see if the market gapped through our protective stop. This is done by comparing our price with the open price of the next bar. If the open price is below our protective stop we then reassign price to the open of the next bar. Since we are exiting a long position, we know there will be profit/loss associated with this trade. We take the price that we got out of the trade and subtract the entry price of the trade. If the price is above the entry price then we have a profit, else we have a loss. We then multiply the difference by the big point value to get the P/L in dollars. We then set the stoppedOut flat to 1 or true. We put the P/L in column 13.

```
If pos = 1 And myLow(i + 1) <= longLiqPrice And longLiqPrice > sellStop Then
    pos = 0
    price = longLiqPrice
    If (myOpen(i + 1) < price) Then price = myOpen(i + 1)
    trdProfit = (price - entryPrice) * bigPointValue
    Cells(displayRow + 1, 13) = trdProfit  'put trade profit in column 13
    stopPrice = 0
    stoppedOut = 1
 End If
```

Just the opposite is done to check for the short side. We check the next days high against the protective stop and make sure the protective stop is closer than the buy reversal stop. We compare the next bars open with the protective stop to check for a gap opening. We also change the pos to 0 for flat and set the stoppedOut flag to 1 or true.

```
If pos = -1 And myHigh(i + 1) >= shortLiqPrice And shortLiqPrice < buyStop Then
    pos = 0
    price = shortLiqPrice
    If (myOpen(i + 1) > price) Then price = myOpen(i + 1)
    trdProfit = (entryPrice - price) * bigPointValue
    Cells(displayRow + 1, 13) = trdProfit ' put trade profit in column 13
    stopPrice = 99999
    stoppedOut = 1
End If
```

Now if we had a position and the buy reversal was closer to the market than the protective stop or we were flat, then we check to see if our buy stop was hit by comparing the next days high with the buy stop. If the high is greater then we assign our price to either the buy stop or the open of the next bar. We also need to check if we closed out an existing position and if so calculate the P/L.

```
If (pos <> 1 And myHigh(i + 1) >= hh And stoppedOut = 0) Then
    price = hh
    If myOpen(i + 1) > price Then price = myOpen(i + 1)
```

```
        longLiqPrice = price - protStop
        If (pos = -1) Then
            trdProfit = (entryPrice - price) * bigPointValue
            Cells(displayRow + 1, 13) = trdProfit ‘ put trade profit in column 13
        End If
        pos = 1
    End If
```

Now we check to see if our sell stop was hit, if we are not already short. Again we calculate our P/L based on if were long coming into the new trade. Also, I forgot to mention, we go ahead and calculate our new protective stop based on our new entry price and put this information into the shortLiqPrice/longLiqPrice variable.

```
    If (pos <> -1 And myLow(i + 1) <= ll And stoppedOut = 0) Then
        price = ll
        If myOpen(myOpen(i + 1)) < price Then price = myOpen(i + 1)
        shortLiqPrice = price + protStop
        If (pos = 1) Then
            trdProfit = (price - entryPrice) * bigPointValue
            Cells(displayRow + 1, 13) = trdProfit ‘put trade profit in column 13
        End If
        pos = -1
    End If
```

We have checked for trades for today and calculated our new positions and any P/L. Now we put our new information into the corresponding columns. Column 10 is assigned our position, Column 11 our entry price and Column 12 our protective stop price.

```
    If Cells(displayRow + 1, 3) <> “” Then
        Cells(displayRow + 1, 10) = pos
        Cells(displayRow + 1, 11) = price
        If (pos = 0) Then Cells(displayRow + 1, 12) = 0
        If (pos = 1) Then Cells(displayRow + 1, 12) = longLiqPrice
        If (pos = -1) Then Cells(displayRow + 1, 12) = shortLiqPrice
        If (pos <> 0) Then entryPrice = price
    End If
```

Done with the day so now let’s increment the data array and cells indices. The keyword loop branches program flow back to the do-while statement and we start a new day.

```
    i = i + 1 ‘main array counter
    displayRow = displayRow + 1
Loop
```

We are now sitting on the last row of data. Cells(displayRow + 1,1) (the date column) is blank therefore signifying the end of the data. Now we have all the information that we need to display the next bar’s orders. If we are flat we put “Buy Tomorrow at” in orderString1 and “SellShort Tomorrow at” in orderString2.

Using Excel’s Visual Basic to Test & Trade Your System - Part 1

We put the respective stop prices in orderPrice1 and orderPrice2. We also put the order information 1 row below the last line of data in columns 2 and 3. The orderStrings and orderPrices will be passed back to the user dialog.

```
If (Cells(displayRow + 1, 1) = “”) Then
    If (pos = 0) Then
        orderString1 = “Buy Tommorrow at : “
        orderString2 = “SellShort Tomorrow at : “
        orderPrice1 = hh
        orderPrice2 = ll
        Cells(displayRow + 1, 2) = “Buy Tomorrow at : “
        Cells(displayRow + 2, 2) = “SellShort Tomorrow at : “
        Cells(displayRow + 1, 3) = hh
        Cells(displayRow + 2, 3) = ll
    End If
```

If we are short then we only put “Buy Tomorrow at : “ in orderString1 and into the row following the last line of data. We then calculate the current OTE and put that into the lastPL variable. We compare the current protective stop to the reversal sell stop. If the protective stop is closer we then change the orderString1 to “Liq Short Tomorrow.” OrderPrice1 is then loaded with either the lowest low value or the protective stop - whichever is closer.

```
    If (pos = -1) Then
        Cells(displayRow + 1, 2) = “Buy Tomorrow at : “ ‘Put the order in the last row
        orderString1 = “Buy Tomorrow”            ‘Put the order in the order string holder
        If hh <= shortLiqPrice Then
            Cells(displayRow + 1, 3) = hh
        Else
            Cells(displayRow + 1, 3) = shortLiqPrice    ‘Put the order in the last row
            orderString1 = “Liq Short Tomorrow” ‘Put the order in the order string holder
            Cells(displayRow + 1, 2) = “Liq Short Tomorrow”
        End If
        orderPrice1 = Cells(displayRow, 3)
        lastPL = (entryPrice - myClose(i1)) * bigPointValue
    End If
```

If we are long we go through the same process. However, in this case we dealing with selling short or liquidating a long position.

```
    If (pos = 1) Then
        Cells(displayRow + 1, 2) = “SellShort Tomorrow at :”‘Put the order in the last row
        orderString2 = “SellShort Tomorrow”    ‘Put the order in the order string holder
        If ll > longLiqPrice Then
            Cells(displayRow+1, 3) = ll
        Else
            Cells(displayRow+1, 3) = longLiqPrice       ‘Put the order in the last row
            orderString2 = “Liq Long Tomorrow”  ‘Put the order in the order string holder
```

```
             Cells(displayRow+1, 2) = “Liq Long Tomorrow”‘
         End If
         orderPrice2 = Cells(displayRow+1, 3)
         lastPL = (myClose(i) - entryPrice) * bigPointValue
     End If
  End If
End Sub
```

Well that’s it for the first part. Keep going through the logic until you understand it. If you do come across something you don’t understand, drop me an email at gpruitt@futurestruth.com. This code could be streamlined a lot more than what is presented. I wanted to be as elaborate as possible in my code to help with readability. Also I have not yet discussed the development of the Dynamic BreakOut dialog nor the code that goes along with it. I just wanted to present the software and an overview of how I put the program together. Feel free to take the code and modify or expand it. In the next issue I will delve into how the dialog box was designed and the associated VB code. We will also get closer to developing a generic back tester.

Making Visual Basic For Excel Test Your Trading Ideas - Part 2

In the last installment of George's Corner we programmed an order generator for the Dynamic Break Out system. In this Corner, we will use the concepts from that program and develop a universal back tester. This back tester will be a beta version (1.0) and will simply be the backbone of what will evolve into a somewhat sophisticated software application. The order generator program was designed specifically for the DBS system. However, changing it to generate signals for a different algorithm would not be that difficult. In fact we will be using a good portion of that code in our "System Tester Vers. 1.0".

Download the code used in George's Corner at ftp://www.FuturesTruth.com/pub/SystemTester.xls.

I have programmed all of the subroutines into a single module to help reduce complexity. Before we start, I suggest downloading the code, printing it out and going through it as you read this George's Corner. You can download the code at ftp://www.FuturesTruth.com/pub/SystemTester.xls. The **Dim** statements at the top of the program are outside any of the subroutines and their associated headers. By dimensioning these variables here, we are forcing them to have global scope. This programming term simply means that all of the different subroutines in the module will have access to the values that are held in these variables. If I am in the Buy subroutine and I need to know what my current market position is, I simply evaluate the marketPosition variable. If I hadn't put these variables outside all of the subroutines and made them global, then they would not be accessible to all of the subroutines. They would only be accessible to the subroutine in which they were defined and to those subroutines in which the variable is passed as an argument. Since this new programming consists of multiple subroutines, unlike our order generator, we needed certain variables visible to most of the subroutines. Making variables global does come at a cost of memory, but with this little program it won't make any difference. You may recognize some of the variables from the Dynamic Break Out 1.1 code. The data arrays declarations are also outside the main subroutine module. You will also notice some global variables like **totProfit, maxDD, perCentWins, numTrades** and others. These are the variables that we will use to load some simple performance metrics.

```
Dim stdArr(29), myDate(500), myHigh(500), myLow(500), myOpen(500), myClose(500) As
Double
Dim equityStream(500) As Double
Dim myVal1, myVal2, myVal3, myVal4, myVal5 As Double
Dim marketPosition, entryPrice, executionCount, entryBar As Integer
Dim myBigPointValue, myMinTick, rampUp As Double
Dim totProfit, maxDD, perCentWins, numTrades, numWins, numLosses, commsn As Double
```

There are a total of six different subroutines. This six subroutines make up the System Tester program. At this point you maybe asking why not simply use one huge subroutine like we did in the order generator instead of many smaller ones? This question can be answered with one word, modularity. Modularity cuts down on redundant code. Every time we close out a trade we need to calculate the profit or loss, number of wins and losses, cumulative profit and total trades. There are four different times we must do this. These times are when we:

1.) liquidate long position
2.) liquidate a short position
3.) reverse a long position and go short
4.) reverse a short position and go long

In our code we would need to copy/paste the same code four different times to calculate the performance metrics of closing out a trade. Or we could simply create one **CalcTradeResults** subroutine and call it four times. I like the latter a whole bunch more.

The first subroutine that we come across is the main subroutine. It is called **SystemTester** and as in similar fashion as the **DBSSub** subroutine we pass information concerning the market being tested to it. Since we are using multiple subroutines and global variables we don't need to pass as many arguments as we did in the **DBSSub**. Here we are simply passing the **BigPointValue** and **MinTick** of the market being tested and the amount of rampUp data needed to generate a trade signal.

```
Sub SystemTester(BigPointValue, MinTick, rampUp)
```

The next few lines initiate some of the global variables that we will use later on in the program. You will probably recognize the Do While loop where we are reading the data from the worksheet into our data arrays if you are following along with a print out of the logic.

```
stp = 1
lmt = 2
mkt = 3
executionCount = 1
marketPosition = 0
myBigPointValue = BigPointValue
myMinTick = MinTick
commsn = 100 / myBigPointValue
totProfit = 0
maxEquityHigh = -9999999
numTrades = 0
```

We then encounter another loop that runs through each day of data. This is where we would program our trading system. I have created a very simple system to help us test and debug our code. This system buys tomorrow at today's high on a stop if the today's close is greater than the previous day's close. It sells tomorrow at today's low if the close of today is less than the prior day close. A $2000 protective stop/ profit objective is also utilized. Yeah - you can probably already tell this is going to be a big loser. But right now we don't care, we just need something that generates a bunch of trades so we can put our software to the test. Some of these subroutine calls have very similar names as other testing platforms out there. This was by design. I have always liked the keywords **Buy** to initiate a long position, **Sell** to initiate a short position, **ExitLong** to exit a long position and **ExitShort** to exit a short position. I am not to keen on how TraderStation™ has changed these names in there version 7 and above. You can tell who

Making Visual Basic For Excel Test Your Trading Ideas - Part 2

they wanted their audience to be. SellShort and BuyToCover - gimme a break. This little snipped of code tells us how to reference prior days **(myClose(i) > myClose(i-1))** and the arguments we need to pass to our subroutines.

```
If myClose(i) > myClose(i - 1) Then Call Buy("Go-Long", myHigh(i), stp, i)
If myClose(i) < myClose(i - 1) Then Call Sell("Go-Short", myLow(i), stp, i)

Call ExitLong("Long-Liq", entryPrice - 2000 / myBigPointValue, stp, i)
Call ExitShort("Short-Liq", entryPrice + 2000 / myBigPointValue, stp, i)
Call ExitLong("Long-Prof", entryPrice + 2000 / myBigPointValue, lmt, i)
Call ExitShort("Short-Prof", entryPrice - 2000 / myBigPointValue, lmt, i)
```

If you want to compare today's high with the high 5 days ago, you would simply subtract five from our array index (i). For example lets say we want to check to see if today's high is greater than the high 5 bars back. The code that would do this is: `if myHigh(i) > myHigh(i-5) then do something.` When we call our Buy/Sell and ExitLong/Short we must do so with information concerning where we want to enter or exit.

The Buy/Sell subroutines need to have the following information passed to it in this exact order:

entry signal name in double quotes - any name can be used
buy/sell level
order type - stp, lmt or mkt - use these variable names
current bar - simply pass the current value of i

If you don't pass the correct information in the correct order, then the program simply will not work.

The ExitLong/Sell subroutines need to have basically same information in the same order:

exit signal name in double quotes - any name can be used
buy/sell level
order type - stp, lmt or mkt - use these variable names
current bar - simply pass the current value of i

If you can manipulate the data arrays by changing their indices and call a Buy/Short and ExitLong/Short subroutine then you can easily test your trading ideas. I have done the hard work for you. However, since you may want to know what is going on behind the scenes, and I hope you do, we will go over each of these subroutines. Again remember this is simply a backbone and as of yet not very sophisticated. You could take the ball and run with this little bit of code and make something sophisticated, or simply wait until the next George's Corner. Right now we haven't programmed any indicators such as a moving average, Bollinger band, Keltner channel, rsi, etc.,. We may do this in the next issue. Also we are still working with the same old data as we did with the order generator program and I haven't programmed any fancy looking reports. They may also be programmed later.

Let's first take a look at the Buy subroutine and see what it does:

Making Visual Basic For Excel Test Your Trading Ideas - Part 2

```
Sub Buy(sigName, trdPrice, orderType, index)
Dim tradeProf, price As Double

price = 0
```

The next snippet of code differentiates between order types and checks the price levels against the order prices. If the order is a limit and we are looking to buy then we check the low price of day against the order price. If the low of the day is less than the price we fill the order at the order price. We then check against the open price to see if there was a gap opening below our limit. If there was we then fill the order at the open price. We look at the high of the day to check our buy stop level. If the high exceeds the stop level, then we know we were filled. We first assume we were filled at our stop level and then we check the open of the day to see if the market gapped up through our price. The other type of order is simply a market order and we fill that at the open price.

```
If (marketPosition <> 1) Then
        Select Case orderType
                Case 1
                If myHigh(index + 1) >= trdPrice Then           'Limit Order
                        price = trdPrice
                        If (myOpen(index + 1) > trdPrice) Then price = myOpen(index + 1)
                End If
                Case 2
                If myLow(index + 1) <= trdPrice Then            'Stop Order
                        price = trdPrice
                        If (myOpen(index + 1) < trdPrice) Then price = myOpen(index + 1)
                End If
                Case 3
                        price = myOpen(index + 1)               'Market Order
        End Select
```

If price <> 0 then we know that we were filled. See above where we set price equal to 0. The only way our price changes is if we in fact did get filled. If we get filled we then select sheet2 in our workbook and fill the cells with the trade information. We put date of the trade in column 1, the name of the signal in column 2 , the entry price in column 3 and the profit or loss, if any, in column 4.

```
If (price <> 0) Then
        Sheets("Sheet2").Select
        entryBar = index + 1
        numTrades = numTrades + 1
        executionCount = executionCount + 1
        Cells(executionCount, 1) = myDate(index + 1)
        Cells(executionCount, 2) = sigName
```

See how we put the values in the different columns.

Here we call the **CalcTradeResults** subroutine to calculate the results of our entry. We only do this if we are currently in a short position.

```
    If (marketPosition = -1) Then
           Call CalcTradeResults(marketPosition, entryPrice, price, tradeProf)
    End If
    entryPrice = price
    Cells(executionCount, 4) = tradeProf * myBigPointValue
    Cells(executionCount, 3) = entryPrice
    marketPosition = 1  Assign our entry price and change marketPosition to 1 - long.
   End If
End If
```

The **Sell** subroutine does basically the same thing, but we check our order price levels differently when we are looking to sell. If we are looking to go short and we wish to enter on a limit order, we compare the high of the day against our limit price. If the high of the day exceeds the sell limit then we know we were filled. We don't know, until we check, if we were filled at the opening on a gap up. Sell stops are below the market so we check the low and open of the day against the stop price to see if we should have been filled. This is basically the only difference between the buy and sell subroutines.

```
If (marketPosition <> -1) Then
   Select Case orderType
      Case 1
         If myLow(index + 1) <= trdPrice Then
            price = trdPrice
            If (myOpen(index + 1) < trdPrice) Then price = myOpen(index + 1)
         End If
      Case 2
         If myHigh(index + 1) >= trdPrice Then
            price = trdPrice
            If (myOpen(index + 1) > trdPrice) Then price = myOpen(index + 1)
         End If
      Case 3
         price = myOpen(index + 1)
   End Select
```

The **ExitLong** subroutine isn't really that different than the **Sell** Subroutine. Again we must differentiate between order types and put the trade information into the different columns. Here we do change our market position to 0 or flat. In fact, if we wanted to streamline our code, we could combine the **ExitLong** and **Sell** subroutines into one subroutine. I made the separate routine to mimic the subroutine calls of other testing platforms. The same goes for the **ExitShort** subroutine.

The last subroutine that we need to talk about is **CalcTradeResults()**. This subroutine is called from each of the four trade entry/exit routines and keeps track of the performance of the trading system. The only information this routine needs to know is the current position, entry price and exit price. It takes this information, processes it, updates the global performance variables and also returns the current trades profit or loss in the **tradeProf** variable. **TradeProf** is passed to the subroutine and is modified and passed back to the calling program. All variables are passed by reference to subroutines or functions in Visual Basic. This simply means the values of these variables can be changed in the subroutine and these changes are reflected in the calling subroutine.

Making Visual Basic For Excel Test Your Trading Ideas - Part 2

```
Sub CalcTradeResults(pos, entryPrice, exitPrice, tradeProf)

        If pos = 0 Then Return       'Return and make no changes if we are flat

        If pos = 1 Then
                tradeProf = exitPrice - entryPrice - commsn
        Else
                tradeProf = entryPrice - exitPrice - commsn
        End If
        If (tradeProf >= 0) Then
                numWins = numWins + 1
        Else
                numLosses = numLosses + 1
        End If
        totProfit = totProfit + tradeProf

        If (totProfit > maxEquityHigh) Then maxEquityHigh = totProfit
        If (maxEquityHigh - totProfit > maxDD) Then maxDD = maxEquityHigh - totProfit

End Sub
```

If we come into this subroutine with a long position we then subtract our exit price from our entry price. If our exit is above our entry we then know we made a profit. If we are short we then subtract our entry price from our exit price, just the opposite of what we did for a long trade. If our entry is above the exit then the short trade was profitable. Once we calculate the trade profit, we then calculate the total overall profit, number of wins, number of losses, and maximum draw down. And that's basically all there is to the System Tester program. You can download the code directly from Futures Truth website and start playing around with it. Remember, to run the program you go under Tools in the Excel Menu, select Macros and highlight the Macro submenu and then select RunSystemTester. You will need to invoke the Visual Basic editor under the tools menu to be able to edit the program. In the next and last installment of developing this system tester we will incorporate some indicators and introduce several different systems and improve on the performance reports.

Making Visual Basic For Excel Test Your Trading Ideas - Part 3

Expanding Our Testing Capabilities With Indicators

Now that we have the core testing engine programmed let's go ahead and program a handful of indicators into subroutines so we can use them in our testing. I got tired of using the same old data that was used in the last two installments, so I have populated the latest SystemTester.xls workbook with November Crude Oil data. I only picked five popular indicators to include in our Visual Basic code, but once you see how they are programmed you should have no problem adding more.

Download the code used in George's Corner at ftp://www.FuturesTruth.com/pub/SystemTester.xls.

One of the easiest indicators to program is the simple moving average so we will start with that one. As you know you can calculate the moving average on different prices. The most popular of course is the close. Since we want to make our indicators as flexible as possible, we should allow the user to pass any price stream (open, high, low or close) to our subroutine and return the moving average of whatever length they choose. Visual Basic is very flexible language and therefore great for the inexperienced programmer. Take a look at the following code to see how simple the process is of adding an indicator to our program.

```
Function Average(dataList, length, index)

Dim i As Integer
Dim sum,  As Double

For i = index - (length - 1) To index
   sum = sum + dataList(i)
Next i
Average = sum / length

End Function
```

Since we are only need one value passed back to the calling subroutine we can simply use a Function. Functions are similar to subroutines in that information is passed back and forth from the calling program and sub program. The main difference between these two subprograms is in how you invoke them. With a subroutine you must use the keyword `Call` prior to the name of the subroutine, whereas a function is accessed by simply using its name as and parameter list. For example, to use the Average function we would use the following snippet of code:

```
myAverage = Average(myClose,14,i)
```

Here we simply assign the 14 day moving average of closing prices to the `myAverage` variable. This was done by using the function name and the parameters necessary to provide the information that is needed. The function header Function Average(dataList, length, index) provides the name of the function (Average) and the list of arguments it needs to do the moving average calculation. I used the variable dataList so that different data arrays could be passed to the function. We could call the function with myClose, myHigh, myLow or myOpen data arrays. Somewhere in the body of the function, the name of the function must be used and assigned the result of a calculation. In the example above we Dim dimension two variables i and sum. These are the only two variables we will need to complete our moving average calculation. Since we will need to look back over n days to sum up the prices, a for loop will be used. The sum variable will act like an accumulator and sum up the prices for the last n days. When the loop terminates the sum will be divided by the number of days that we have looked back and the quotient will be assigned to the name of our function - Average. Thats all there is to this simple indicator. You could use this template to create an exponential or weighted moving average indicator or any other indicator that has just one output. Here is a little more complicated indicator that can be programmed as a function.

```
Function RSI(dataList, length, index)

Dim i As Integer
Dim diff1, diff2, upSum, dnSum

upSum = 0
dnSum = 0

If rsiVal1 = 0 And rsiVal2 = 0 Then     'seed the original RSI Value
   For i = index - (length - 1) To index
      If dataList(i) > dataList(i - 1) Then
         diff1 = dataList(i) - dataList(i - 1)
         upSum = upSum + diff1
      End If
      If dataList(i) < dataList(i - 1) Then
         diff2 = dataList(i - 1) - dataList(i)
         dnSum = dnSum + diff2
      End If
   Next i
   rsiVal1 = upSum / length
   rsiVal2 = dnSum / length
Else
   If dataList(index) > dataList(index - 1) Then
      diff1 = dataList(index) - dataList(index - 1)
      upSum = upSum + diff1
   End If
   If dataList(index) < dataList(index - 1) Then
      diff2 = dataList(index - 1) - dataList(index)
      dnSum = dnSum + diff2
   End If
   rsiVal1 = (rsiVal1 * (length - 1) + upSum) / length
```

```
    rsiVal2 = (rsiVal2 * (length - 1) + dnSum) / length
End If
If rsiVal1 + rsiVal2 <> 0 Then
    RSI = (100 * (rsiVal1)) / (rsiVal1 + rsiVal2)
Else
    RSI = 0
End If

End Function
```

This RSI function is a little more complicated because the indicator initially needs to be calculated with the use of simple moving average. Here we once again pass the data array to the function because the RSI can be calculated on any data stream.

```
upSum = 0
dnSum = 0

If rsiVal1 = 0 And rsiVal2 = 0 Then     'seed the original RSI Value
    For i = index - (length - 1) To index
        If dataList(i) > dataList(i - 1) Then
            diff1 = dataList(i) - dataList(i - 1)
            upSum = upSum + diff1
        End If
        If dataList(i) < dataList(i - 1) Then
            diff2 = dataList(i - 1) - dataList(i)
            dnSum = dnSum + diff2
        End If
    Next i
    rsiVal1 = upSum / length
    rsiVal2 = dnSum / length
Else
```

`rsiVal1` and `rsiVal2` are global parameters (not local to just this function) and if they are equal to zero this must mean we have entered this function for the first time. Once we change these values they will be the same until they are changed again. Just to refresh your memory on the RSI, this indicator first averages the magnitudes between the positive and negative differences between today's close (open, high, low) and yesterday's close. These values will be positive since we are simply looking at the magnitudes of the differences. The RSI is then calculated by dividing the average magnitude of up closes by the sum of the average magnitude of up closes and the average magnitude of down closes. This quotient is then multiplied by 100. This indicator oscillates from 0 to 100. After we initially calculate the RSI (seed), we then use the last `rsiVal1` and `rsiVal2` to calculate the next value of the RSI.

```
  If dataList(index) > dataList(index - 1) Then
        diff1 = dataList(index) - dataList(index - 1)
        upSum = upSum + diff1
    End If
```

Making Visual Basic For Excel Test Your Trading Ideas - Part 3

```
    If dataList(index) < dataList(index - 1) Then
        diff2 = dataList(index - 1) - dataList(index)
        dnSum = dnSum + diff2
    End If
    rsiVal1 = (rsiVal1 * (length - 1) + upSum) / length
    rsiVal2 = (rsiVal2 * (length - 1) + dnSum) / length
```

First we calculate the most recent up or down close and then use a simplified moving average calculation to come up with the most recent RSI value. We take the last value of `rsiVal1` and multiply it my length -1 and then add the `upSum` and then divide this result by the length. This was Welles Wilder's shortcut for calculating averages before the computer era. `RsiVal1` and `rsiVal2` keep their values inside and outside the RSI function because they were declared globally. Hence we can count on them having their last values inside our function. Some times indicators can be more complex and require more than just one value to be calculated. The Bollinger Band indicator is one that outputs two or three different values. The Bollinger Band indicator calculates a moving average and one band above and one band below the average. The distance between the bands and the average is based on the user defined multiple of the standard deviation of the moving average. Most traders like to see one or two standard deviations above/below the moving average line. These bands signify either support or resistance.

```
Sub BollingerBand(dataList, length, numDevs, avg, upBand, dnBand, index)

Dim i As Integer
Dim sum, sum1, myDev As Double

For i = index - (length - 1) To index

    sum = sum + dataList(i)
    sum1 = sum1 + dataList(i) ^ 2

Next i

avg = sum / length

myDev = ((length * sum1 - sum ^ 2) / (length * (length - 1))) ^ 0.5

upBand = avg + myDev * numDevs
dnBand = avg - myDev * numDevs

End Sub
```

Here we use a subroutine instead of a function. We will want three outputs from this subroutine: `avg`, `upBand` and `dnBand`. The subroutine will need to know the price stream, the number of bars to look back and the number of standard deviations. This subroutine uses a for loop to calculate the sum of prices, and the sum of the prices squared. Again we could be using the open, high, low or close data stream in this loop. The `myDev` variable is assigned the standard deviation calculation based on the length or sample size of the data. Once this variable is changed it stays that way until it is changed again, so we can access the value in our main loop. Just for a little more practice in coding these indicators the Stochastic programming is also included in the workbook.

If you run the SystemTesterV1.2 module, the values of these indicators are printed out in columns 8 through 14 on “Sheet1”. This was accomplished by this following code:

```
Sheets(“Sheet1”).Select     ‘Select Sheet1 to access the data
Cells(i + baseRow, 8) = upBand
Cells(i + baseRow, 9) = dnBand
Cells(i + baseRow, 10) = simpleAvg
Cells(i + baseRow, 11) = rsiVal
Cells(i + baseRow, 12) = stoK
Cells(i + baseRow, 13) = stoD

Cells(i + baseRow, 14) = slowD
```

A simple Bollinger Band system was programmed to demonstrate the use of these indicators.

```
If myClose(i) > upBand Then Call Buy(“Go-Long”, myClose(i), mkt, i)
If myClose(i) < dnBand Then Call Sell(“Go-Short”, myClose(i), mkt, i)
if myClose(i) < avg Then Call ExitLong(“LongLiq”,myClose(i),mkt,i)
if myClose(i) > avg Then Call ExitShort(“ShortLiq”,myClose(i),mkt,i)
```

If the close of the day is above the top Bollinger band then we enter the market on the close. I had to change a little bit of code to allow market orders to be executed on the close of the day. This change has been reflected in latest version of the SystemTester. If the close of the day is less than the bottom Bollinger then we enter the market on the close. If we do get into a long position and the market then retraces back and closes below the moving average we then exit our long position. The same works for short positions. We exit short when the market moves back up through the moving average.

In the next installment, we will add the ability to have multiple sheets of data and have the program test all of them in one run. As usual, just go to our website to download SystemTesterV1.2.xls.

If you have any questions on this article or code it contains, feel free to contact George directly at GPruitt@FuturesTruth.com.

Making Visual Basic For Excel Test Your Trading Ideas - Part 4

Putting the Thermostat System Into Our Visual Basic Testing Platform

I developed the Thermostat system back in 2001 and published it in *Building Winning Trading Systems with TradeStation*. This system is a combination of two systems: a short term swing system and a trend follower. The trend following mechanism is simply a 2 standard deviation break out (Bollinger Band) of a 50 day moving average. The swing system is a simple open range breakout. What makes this system different than others is its ability to switch from one system to the other. The function that causes the system to switch modes is the Choppy Market Index. This index tells us what mode the market is currently exhibiting: choppy or trending. This formula is quite simple – we divide the actual distance the market has traveled for the past thirty days by the distance that market has wandered for the past thirty days.

```
Choppy Market Index = ABS(Close – Close(29))/(Highest(High,30)-Lowest(Low,30)) * 100
```

If the distance between today's close and the close of 30 days ago is less than 20% of the distance between the highest high and the lowest low of the past 30 days then I feel the market is demonstrating a swinging motion. On the other hand if the market has resulted in a gain or loss of 20% or more of the highest high – lowest low range, then I feel the market is in a trend mode. If you want to simply work with decimals you can eliminate the 100 multiplier. This I have done in the trading logic.

If the Choppy Market Index (CMI) is less than 20%, then we will utilize the open range break out (ORBO) mechanism. The ORBO also incorporates pattern recognition. While in choppy mode we can have either a buy easier or sell easier day. These types of days are determined by comparing today's closing price to the key of the day (KOD). The KOD or day trader's pivot point is calculated by simply dividing the sum of today's high, low, and close by three (H + L + C)/3. If the close is above the KOD then we have a sell easier day (SAD). In other words the market closed nearer its high and we are expecting a down day tomorrow. If the close is below or equal to the KOD, then we have a buy easier day (BAD). These patterns are pretty self explanatory. On BAD days we will make it easier to buy than sell and, on SAD days sells will be easier than buys. On BAD days we will buy at the open + 50% of the ten day average true range (10ATR) and sell short at the open – 75% of 10ATR. On SAD days we will just do the opposite – buy at the open + 75% of the 10ATR and sell short at the open – 50% of the 10ATR. The weakness with break out systems is that they sometimes get whipsawed in extremely choppy markets. I tried to prevent this by utilizing a Trend Lock Point (TLP). Many times we will have a gap down that continue a down trend and then the market attempts to fill the gap. A pure break out system will be tempted to buy as the market moves up off of the open even though the market is still showing weakness. The TLP attempts to keep the system in synch with the current short term trend. If the buy entry stop is below the three day average of low prices, I move the buy stop up to this point. If the sell stop is above the three day average of high prices, I move the sell stop down to this point. In other words:

```
BuyStop = MAX(BuyStop,Average(Low,3))

SellStop = MIN(SellStop,Average(High,3))
```

If a position is initiated a three 10ATR protective stop is invoked. This stop is rarely hit.

Making Visual Basic For Excel Test Your Trading Ideas - Part 4

If the CMI determines that we are in a trending market, then the system switches to the 50-day Bollinger break out. Long positions are initiated at the upper Bollinger band and short positions are put on when the market moves past the lower Bollinger band. While we are in a trending market the three 10ATR stop is not utilized. Liquidations of positions take place at the 50-day moving average. If we enter a long position at the upper Bollinger band, we then will liquidate once the market moves down past the 50-day moving average. Short positions are liquidated when the market moves above the 50-day moving average.

That's all there is to this system. Since release the system has been very successful even though it did surpass its worse draw down in late 2006. I thought this would be a good system to use as a tutorial for programming in our system tester software.

I have updated the SystemTester to version 1.3. The new version includes the Highest/Lowest functions and I have divided the program into different areas by using comments. You now know where to declare your variables and put your system logic. You can download the ThermostatTest Excel spreadsheet from our website.

Now that we know how the system works, let's see how easy it is to program into our SystemTester. I will only include the necessary logic for the system to help reduce space. First off we will declare the necessary variables that we will need for Thermostat.

```
Dim cmiVal As Double
Dim hh, ll, trendLokBuy, trendLokSell, swingBuyPt, swingSellPt, trendBuyPt, trendSellPt As Integer
Dim choppyPer1, choppyPer2, keyOfDay, protStopAmt As Double
Dim buyEasierDay, sellEasierDay, bollAvg As Integer
```

Many times I don't know ahead of time what variables I will need so I make them up as I go along. You will notice that I have dimensioned some of my variables as Integers and some as Doubles. If I need a variable to have a fractional part, I will declare it as a Double. If I need a toggle, like the `buyEasierday` or `sellEasierDay` or an entry/exit price I will usually dimension as an Integer. You could dimension everything as Double if you like.

After the declaration (dimensioning) of the variables we move into the main trading loop and do the necessary calculations for the Choppy Market Index (CMI).

```
choppyPer1 = 0.5
        choppyPer2 = 0.75

        hh = Highest(myHigh, 30, 0, i)
         ll = Lowest(myLow, 30, 0, i)

        cmiVal = Abs(myClose(i - 29) - myClose(i)) / (hh - ll)
```

Here we assign 0.5 and 0.75 to the first and second choppy percentage variables. We then invoke the Highest and Lowest functions to determine the highest high and lowest low values for the past 30 days.

Making Visual Basic For Excel Test Your Trading Ideas - Part 4

The cmiVal is calculated by dividing the absolute value of the close 30 days ago minus today's close by the difference between the highest high and the lowest low for the past thirty days.

```
trendLokBuy = Average(myLow, 3, i)
           trendLokSell = Average(myHigh, 3, i)

          buyEasierDay = 0
          sellEasierDay = 0

           keyOfDay = (myClose(i) + myHigh(i) + myLow(i)) / 3

           If (myClose(i) > keyOfDay) Then sellEasierDay = 1
           If (myClose(i) <= keyOfDay) Then buyEasierDay = 1
```

This snippet of code calculates the trendLokBuy and trendLokSell points. These values are determined by calling the Average function using the myLow and myHigh data arrays. The next value to be calculated is the Key of the Day. This is calculated by dividing the sum of the day's high, low and close by three. We use this value to determine if the next day is going to be a buyEasierDay or a sellEasierDay. If today's close is greater than the Key of the Day, then we will setup for a sellEasierDay tomorrow. Conversely, if the market closes at or below the Key of the Day, then we will set up for a buyEasierDay.

```
If (buyEasierDay = 1) Then
                    swingBuyPt = myOpen(i + 1) + Average(myTrueRange, 10, i) * choppyPer1
                    swingSellPt = myOpen(i + 1) - Average(myTrueRange, 10, i) * choppyPer2
          End If
          If (sellEasierDay = 1) Then
                    swingBuyPt = myOpen(i + 1) + Average(myTrueRange, 10, i) * choppyPer2
                    swingSellPt = myOpen(i + 1) - Average(myTrueRange, 10, i) * choppyPer1
          End If
```

The buyEasier/sellEasier day only applies to the short term swing system so we can go ahead and calculate the swingBuy/swingSell points once we determine what type of day has set up. If we have a buyEasierDay, then the buy stop will be the Open of tomorrow plus 50% of the ten day average true range. The sell stop will be calculated similarly but we will subtract 75% of the ten day average true range. The buy and sell stops are calculated in the same manner on sellEasierDays; we simply flip the 50% and 75%.

```
Call BollingerBand(myClose, 50, 2, bollAvg, trendBuyPt, trendSellPt, i)

           If (cmiVal < 0.2) Then
                     If marketPosition <> 1 Then Call Buy("SwingBuy", swingBuyPt, stp, i)
                     If marketPosition <> -1 Then Call Sell("SwingSell", swingSellPt, stp, i)
           Else

                    If marketPosition <> 1 Then Call Buy("TrendBuy", trendBuyPt, stp, i
 If marketPosition <> -1 Then Call Sell("TrendSell", trendSellPt, stp, i)

          End If
```

Making Visual Basic For Excel Test Your Trading Ideas - Part 4

Once we calculate the Bollinger Bands we can then start to place the orders. The BollingerBand subroutine assigns the upper band to the trendBuyPt and the lower band to the trendSellPt. So, we now have our trendBuy, trendSell, swingBuy, and swingSell price points. All we need to do now is figure out which ones of these needs to be placed. We only place the swing stop orders in choppy market mode and this occurs when the cmiVal drops below 0.2. If the cmiVal is greater than or equal to 20% we then place the trending stop orders.

After the entry orders are placed we need to place our protective stops.

```
protStopAmt = 3 * Average(myTrueRange, 10, i)
         If marketPosition = 1 Then
                If signalName = “TrendBuy” Then Call ExitLong(“TrendLongEx”, bollAvg, stp, i)
                If signalName = “SwingBuy” Then Call ExitLong(“SwingLongEx”, entryPrice - protS-
topAmt, stp, i)
        End If

        If marketPositon = -1 Then
                If signalName = “TrendSell” Then Call ExitShort(“TrendShortEx”, bollAvg, stp, i)
                If signalName = “SwingSell” Then Call ExitShort(“SwingShortEx”, entryPrice +
protStopAmt, stp, i)
        End If
```

Notice how we tied our exit signals with our entry signals. If we have a long position and the signal name that put us into that trade is a “TrendBuy” then we only exit with the “TrendLongEx” signal. The “SwingBuy/Sell” entries are tied to the “SwingLong/ShortEx” signals.

I have included two years of continuous crude oil in this spreadsheet that you can test on. Please play around with the logic and plug in your own system and see if you can come up with a good system. Two things that I want to change in the next revision, which will increase accuracy, are:

1.) Create a subroutine that determines which order is closest to the market and execute only that one. Right now the system looks at the entries first and if the order is hit it will automatically execute, even though the liquidation order might be closer.

2.) Allow multiple entries on the same bar if we can determine with a high level of accuracy which occurred first – the high or the low of the bar.

Also, here are the codes for the Highest and Lowest functions:

```
Function Highest(dataList, length, offset, index)
Dim i As Integer
Dim tempHH As Integer

        tempHH = 0
        For i = index - (length - 1 + offset) To index
                If (dataList(i) > tempHH) Then tempHH = myHigh(i)
         Next i
        Highest = tempHH
```

```
End Function

Function Lowest(dataList, length, offset, index)
Dim i As Integer
Dim tempLL As Double

          tempLL = 999999
         For i = index - (length - 1 + offset) To index
                  If (dataList(i) < tempLL) Then tempLL = myLow(i)
          Next i
          Lowest = tempLL

End Function
```

I will repeat the core logic of the Thermostat system so that you won't need to extract it from the Visual Basic code. Here it is in its entirety. I would make sure you fully grasped this before tweaking or programming your own system.

```
Do While i < numRecords

    i = i + 1

    choppyPer1 = 0.5
    choppyPer2 = 0.75

    hh = Highest(myHigh, 30, 0, i)
    ll = Lowest(myLow, 30, 0, i)

    cmiVal = Abs(myClose(i - 29) - myClose(i)) / (hh - ll)

    trendLokBuy = Average(myLow, 3, i)
    trendLokSell = Average(myHigh, 3, i)

    buyEasierDay = 0
    sellEasierDay = 0

    keyOfDay = (myClose(i) + myHigh(i) + myLow(i)) / 3

    If (myClose(i) > keyOfDay) Then sellEasierDay = 1
    If (myClose(i) <= keyOfDay) Then buyEasierDay = 1


    If (buyEasierDay = 1) Then
        swingBuyPt = myOpen(i + 1) + Average(myTrueRange, 10, i) * choppyPer1
        swingSellPt = myOpen(i + 1) - Average(myTrueRange, 10, i) * choppyPer2
    End If
    If (sellEasierDay = 1) Then
        swingBuyPt = myOpen(i + 1) + Average(myTrueRange, 10, i) * choppyPer2
        swingSellPt = myOpen(i + 1) - Average(myTrueRange, 10, i) * choppyPer1
    End If
    Call BollingerBand(myClose, 50, 2, bollAvg, trendBuyPt, trendSellPt, i)

    If (cmiVal < 0.2) Then
        If marketPosition <> 1 Then Call Buy("SwingBuy", swingBuyPt, stp, i)
        If marketPosition <> -1 Then Call Sell("SwingSell", swingSellPt, stp, i)
    Else

        If marketPosition <> 1 Then Call Buy("TrendBuy", trendBuyPt, stp, i)
        If marketPosition <> -1 Then Call Sell("TrendSell", trendSellPt, stp, i)

    End If

    protStopAmt = 3 * Average(myTrueRange, 10, i)
    If marketPosition = 1 Then
            If signalName = "TrendBuy" Then Call ExitLong("TrendLongEx", bollAvg, stp, i)
            If signalName = "SwingBuy" Then Call ExitLong("SwingLongEx", entryPrice - protStopAmt, stp, i)
    End If

    If marketPositon = -1 Then
            If signalName = "TrendSell" Then Call ExitShort("TrendShortEx", bollAvg, stp, i)
            If signalName = "SwingSell" Then Call ExitShort("SwingShortEx", entryPrice + protStopAmt, stp, i)
    End If

'---------- End of System Logic - Do Not Change Next 2 lines -----------

    If (totProfit > maxEquityHigh) Then maxEquityHigh = totProfit
    If (maxEquityHigh - totProfit > maxDD) Then maxDD = maxEquityHigh - totProfit

'----------------------------------------------------------------------

Loop
```

Download the code used in George's Corner at ftp://www.FuturesTruth.com/pub/Thermostat.xls.

If you have any questions on this article or code it contains, feel free to contact George directly at George@FuturesTruth.com.

Miscellaneous Ramblings

Could've , Would've , Should've . . .

Long side only commodity trading has been the hot topic for a few years now. Jim Rogers book of a few years ago, *Hot Commodities*, either got this ball rolling or helped it along considerably. Many professional money managers offer this type of product to their clients and even though they may have gone through some draw down, the end result (especially for the past few months) has been quite good. I wanted to see what the results of this type of trading would look like over the past 22 years so I used the Aberration system as my guinea pig. I forced the system to just take the long signals and skip the shorts. Long trades were liquidated at their normal exits. The following table shows performance from the long side only on twelve different commodity markets.

```
                  Avg      Max    in Last 12mn  Trds          %       %Gain
                  $PL   $PL/Yr   DrawDn     $PL  DrawDn   /Yr %Wins  TIM  W:L /Mr+DD
Crude Oil       45830     2075    31780   10680   12120     4  52.3   46  1.5    5.9
Heating Oil     76390     3459    23906   12466   12818     4  53.7   43  1.9   13.3
Natural Gas     84530     4762    34940   -4730    6300     4  43.8   36  1.7   11.6
Soybeans        22795     1032    38395   14095    6915     3  41.3   33  1.3    2.6
Wheat              75        3    18013    7350   14725     2  39.2   26  1.0    0.0
Cotton          31315     1418    18485   -5175    7630     3  42.4   28  1.7    7.3
Rough Rice     -10000     -511    23274    5174    1160     3  37.9   27   .8   -2.0
Corn            10863      492    12863    4075    3450     3  39.0   29  1.4    3.7
Sugar            6978      316     8434    2184    1142     3  45.0   33  1.2    3.5
Copper          66238     2999    22813   -5238   17063     3  43.4   38  2.1   12.0
Pork Bellies    -9048     -410    25220   -1860    3840     3  37.1   28   .9   -1.5
Live Cattle      5360      243    12412   -2112    4760     5  41.0   42  1.1    1.9
```

```
                    Net      Max                # of    % Avg.Mrgn           %Gain
                    $PL   DrawDn      Date    Trades  TIM     Reqd          /Mr+DD
Last  6 Months    21051    37216 on 20080123      26  100     8881            76.5
Last 12 Months    35119    37216 on 20080123      41  100     7216            63.8
Average / Year    14984    26195 Avg. Hi  22      38   95     7555            31.5
Full Run TOTAL   330907    72526 on 20050721     834   95     7555            16.0
```

The results look great except for the maximum draw down. What if we simply bought and held the commodities for this same time period? Would the old buy and hold out perform the long only Aberration approach? The next table holds the answer.

```
                Total      Avg      Max    in Last 12mn  Trds          %       %Gain
                  $PL   $PL/Yr   DrawDn     $PL  DrawDn   /Yr %Wins  TIM  W:L /Mr+DD
Crude Oil       77510     3510    34700   27660   12120     6  54.9  100  1.5    9.2
Heating Oil     73626     3334    59816   33852   13327     6  54.1  100  1.4    5.4
Natural Gas    -28890    -1628   167600  -16460   37640     6  47.7  100   .9    -.9
Soybeans        17810      806    48195   23845    6600     6  49.6  100  1.1    1.6
```

```
Wheat            -3875      -175     44563    16675    10225      5  43.2  100  1.0     -.4
Cotton           -7190      -326     66895      955     5815      5  47.7  100  1.0     -.5
Rough Rice      -30710     -1568     38650     4534     2810      6  38.1  100   .7    -3.9
Corn            -19950      -903     31738     1925     7963      5  38.7  100   .7    -2.8
Sugar             2184        99     13978      157     3024      4  49.4  100  1.0      .7
Copper           95388      4319     35475     1000    22738      5  55.9  100  1.9    11.5
Pork Bellies      6464       293     49304    -5232    12628      5  46.8  100  1.0      .6
Live Cattle      23004      1042     14892    -4332     8140      6  55.6  100  1.3     6.8
_____________________________________________________________________________________________
                    Net       Max                   # of    % Avg.Mrgn            %Gain
                    $PL    DrawDn        Date     Trades  TIM      Reqd            /Mr+
Last   6 Months   60426     38586 on 20080123        37  100     22945           196.4
Last 12 Months    96059     40402 on 20070821        74  100     22945           151.6
Average / Year     9270     45903 Avg. Hi   22       63  100     21525            13.5
Full Run TOTAL   204710    198067 on 20070111      1401  100     21525             4.2
_____________________________________________________________________________________________
```

Buy and hold not only made less money but it almost tripled the maximum draw down. Again, a systematic approach out performs the simple buy and hold. A calculated protective stop must be built in to the system to help with the large retracements that occur in the commodity markets.

Continuous Versus Individual Contract Data

Recently while doing some testing on different trading approaches, I noticed differences in performance numbers when using continuous data versus individual contract data. I used the exact system on the two data formats and observed considerable differences. When you place a trade you place it in the actual market, so you would think testing on actual data would be the most accurate. In some cases this is true. However, there are two important elements that make this difficult if not impossible. The first problem with back testing on actual contract data is that today's popular software can't handle the data management of rolling from one contract to another. The software would have to trade, for example, the March contract up until the rollover date, exit the position, unload the March contract, load the June contract, calculate indicators/patterns on the new June contract and then initiate the position in the new contract. The software could potentially do this twelve times a year (energies trade every month). The software would also have to keep track of the P/L and the draw down from these rollover trades. Our own Excalibur software somewhat does just this. The limitation of Excalibur is that it forces rollovers on the last day of the month prior to the expiration month and many times this is not the right time to rollover. The stock indices continue trading the old contract for a week or more in the expiring month. So it turns out Excalibur is trading a contract that is not yet the top step for a week or so. In a perfect world with perfect software, rollovers would occur based on volume and open interest and not a static date. This type of data management wouldn't be that difficult to program and why it hasn't yet been done is a mystery to me. We have the data for all of the contracts so why can't a program just simply preprocess the data and create a database of front month contracts and keep track of the rollover data for each one. Maybe one day someone will do this at the retail level, because I am sure large and wealthy CTAs have this type of software at their disposal.

The second problem occurs because there is very little overlap data between the expiring and new contract. In the example above I outlined how software would unload the old contract data, load the new contract data and recalculate the trade signals/indicators on the new data. Let's say you are using a 200-day moving average and you load the new contract data into memory to do your calculation, you will notice rather quickly that there isn't 200 days worth of data to do your calculations. The new contract simply does not have that large amount of data because no one traded that far back in time. And if there is data going that far back, it probably is very thin and is not a good representation of how that particular market had been trading. So we would need to add one more job to our dream software; after unloading the March contract and loading the June contract, we would need to create a synthetic continuous contract going back as far as necessary based off the last few contracts of data so we could calculate the signals/indicators. In other words, the software would build continuous data dynamically all the while trading the actual contract data. This may sound highly complex, but it could still be done.

Or we could just simply use synthetic data for the entire test period and ignore rollovers altogether. This is what 90% or more of system testers do. They understand that their results may not exactly match what would have occurred, but at least they can develop an expectation of performance.

I tested two systems on continuous and actual contract data. The first system was a simple 20-day Donchian break out – pure stop and reverse. The second system was a little more complex: buy orders were placed at 20% of the 10-day ATR above the open when the market closed below its previous day's close but above the close three days prior and sell orders were place 20% of the 10-day ATR below the open when the market closed above its previous day's close but below the close three days prior. The two systems are symmetric; buy orders are just the opposite of the sell orders. The second system is also a pure stop and reverse.

The following table shows the performance of the Donchian system. Individual contract data was used in the first test and continuous in the second.

Individual Test

```
                 Total      Avg      Max    in Last 12mn  Trds               %Gain
                   $PL   $PL/Yr   DrawDn     $PL  DrawDn   /Yr %Wins  TIM  W:L /Mr+DD
US Bonds         49050     2221    26050    5450    9420    11  41.7  100  1.2    7.7
Euro Curr-DM    124688     5668    49025    9975    7638    11  42.3  100  1.4   11.0
Cotton           19740      894    36250    5085    3680    13  36.2  100  1.1    2.4
Japanese Yen     92925     4208    35163    9300    7788    11  43.8  100  1.3   11.3
Soybeans          2250      102    30950    9285   13790    14  38.8  100  1.0     .3
```

```
                    Net      Max                 # of     % Avg.Mrgn             %Gain
                    $PL   DrawDn        Date   Trades  TIM     Reqd            /Mr+DD
Last  6 Months    20520    15765 on 20080110       32  100     9710             161.1
Last 12 Months    32650    15765 on 20080110       62  100     9710             128.2
Average / Year    13071    25235 Avg. Hi  22       60  100     9707              37.4
Full Run TOTAL   288648    62772 on 19941114     1323  100     9707              18.0
```

Continuous Test

```
                 Total      Avg      Max    in Last 12mn  Trds          %       %Gain
                   $PL   $PL/Yr   DrawDn     $PL  DrawDn   /Yr %Wins  TIM  W:L /Mr+DD
US Bonds K       46220     2093    23340    5260    9610     7  39.5  100  1.3    8.0
Euro Curr K     136725     6409    49988    8825   10075     7  43.0  100  1.5   12.2
Cotton K         45715     2070    25000    6635    3625     7  37.9  100  1.3    8.0
Japanese K       93888     4252    33150    9613    7750     7  39.9  100  1.4   12.0
Soybeans K       -1050      -48    31830    9910   13125     8  35.3  100  1.0    -.1
```

Miscellaneous Ramblings

```
                     Net      Max                    # of    % Avg.Mrgn          %Gain
                     $PL   DrawDn       Date       Trades  TIM      Reqd          Mr+DD
Last  6 Months     19650    18220 on 20080110        21  100      9710          140.7
Last 12 Months     33863    18220 on 20080110        37  100      9710          121.2
Average / Year     14558    25040 Avg. Hi  22        36  100      9709           41.9
Full Run TOTAL    321498    65835 on 19941114       802  100      9709           19.3
```

The results are not that dissimilar and the differences can be attributed to the execution costs associated with rollovers. The system rolled 506 times and at a cost of $75 that would equal to $37,950 and that is pretty darn close to the difference in the results.

Now for the short term Open Range Break Out with Pattern Recognition system.

Individual Test

```
                   Total       Avg      Max     in Last 12mn  Trds            %        %Gain
                     $PL    $PL/Yr   DrawDn      $PL  DrawDn   /Yr %Wins  TIM  W:L /Mr+DD
US Bonds          -13860      -628    97340    11490    8960    29  45.0  100  1.0     -.6
Euro Curr-DM      -40613     -1846    74800   -11325   24788    30  43.1  100   .9    -2.4
Cotton            169330      7668     9235     9715    3815    30  47.2  100  1.8    74.9
Japanese Yen       28250      1279    50150   -12913   18300    29  45.4  100  1.1     2.4
Soybeans             -35        -2    32095     7610   11580    33  39.8  100  1.0     0.0
```

```
                     Net      Max                    # of    % Avg.Mrgn          %Gain
                     $PL   DrawDn       Date       Trades  TIM      Reqd         /Mr+DD
Last  6 Months    -27305    40005 on 20080124        81  100      9710         -109.8
Last 12 Months      3400    40005 on 20080124       139  100      9710            6.8
Average / Year      6479    21941 Avg. Hi  22       150  100      9702           20.5
Full Run TOTAL    143073    59722 on 20010509      3302  100      9702            9.3
```

Continuous Test

```
                    Total       Avg      Max     in Last 12mn  Trds            %        %Gain
                      $PL    $PL/Yr   DrawDn      $PL  DrawDn   /Yr %Wins  TIM  W:L /Mr+DD
US Bonds K         -21050      -953    86680     8390    8630    24  42.6  100   .9    -1.1
Euro Curr K        -59750     -2801    89488   -12813   27000    26  43.6  100   .9    -3.0
Cotton K            72155      3267    16625     7295    3765    26  45.8  100  1.3    18.5
Japanese K          26988      1222    46513    -7825   12863    25  44.1  100  1.1     2.5
Soybeans K         -63290     -2866    88730     6260    8640    29  36.6  100   .8    -3.2
```

```
                     Net      Max                    # of    % Avg.Mrgn          %Gain
                     $PL   DrawDn       Date       Trades  TIM      Reqd         /Mr+DD
Last  6 Months    -16120    45093 on 20080129        66  100      9710          -58.8
Last 12 Months      -533    46838 on 20070212       116  100      9710            -.9
Average / Year     -2036    11136 Avg. Hi  22       129  100      9700           -9.8
Full Run TOTAL    -44953   130731 on 20030819      2840  100      9700           -1.4
```

As you can see from the results the difference cannot be simply attributed to rollover costs. The financials and currencies are reasonably (term used loosely) close, but the cotton and beans are way off. The smoothing process that is used to create continuous contracts undoubtedly has an effect on the relationships of recent closing prices and potentially on the range of prices themselves. We can see if the smooth ing process has affected the ranges by simply eliminating the pattern from our test. The following tables reflect a simple 20% ATR Open Range Break Out test.

Individual Test

```
                Total      Avg      Max     in Last 12mn  Trds          %     %Gain
                  $PL   $PL/Yr   DrawDn      $PL  DrawDn   /Yr %Wins  TIM  W:L /Mr+DD
US Bonds      1322620    59892     3170    60600    1250   125  67.8  100  6.1 1020.3
Euro Curr-DM  1388450    63111     9625    47650    4888   128  62.0  100  3.3  520.5
Cotton         927860    42016     3535    31370    2520   127  64.1  100  5.5  926.5
Japanese Yen   765825    34679    10413    33625    4025   121  57.1  100  2.3  275.8
Soybeans       677730    30690     6275    50245    3240   133  62.5  100  3.8  402.5
```

```
                    Net      Max                # of    % Avg.Mrgn            %Gain
                    $PL   DrawDn        Date  Trades  TIM     Reqd           /Mr+DD
Last  6 Months   129030     2855 on 20080108     306  100     9710           2053.8
Last 12 Months   221155     2855 on 20080108     626  100     9710           1760.1
Average / Year   230150     5416 Avg. Hi  22     634  100     9710           1521.6
Full Run TOTAL  5082480     8157 on 19990216   14001  100     9710           1288.1
```

Continuous Test

```
                Total      Avg      Max     in Last 12mn  Trds          %     %Gain
                  $PL   $PL/Yr   DrawDn      $PL  DrawDn   /Yr %Wins  TIM  W:L /Mr+DD
US Bonds K    1148450    52005     2680    56560    1550   122  67.3  100  5.8  966.6
Euro Curr K   1312138    61506    21825    40225    4738   124  61.8  100  3.2  252.9
Cotton K       759615    34398     3370    29705    2750   124  64.0  100  4.6  787.1
Japanese K     759963    34413    12150    29875    4425   118  56.6  100  2.4  240.5
Soybeans K     657945    29794     5585    53915    3080   128  61.8  100  3.7  429.6
```

```
                    Net      Max                # of    % Avg.Mrgn            %Gain
                    $PL   DrawDn        Date  Trades  TIM     Reqd           /Mr+DD
Last  6 Months   130940     2778 on 20070809     303  100     9710           2097.1
Last 12 Months   207973     2778 on 20070809     613  100     9710           1665.4
Average / Year   210028     6593 Avg. Hi  22     611  100     9710           1288.3
Full Run TOTAL  4638110    19377 on 20001002   13498  100     9710            722.1
```

Wait! Before you get carried away with the results I need to qualify this test. To do an accurate test I had to cheat a little bit. This open range break out system only buys today when today's close is above the open and only sells when today's close is lower than the open. I am looking at today's close before taking action today, which is impossible. By doing this I force the system to only have one trade a day and the systems should be buying and selling on the exact same days. The number of trades is almost the same once rollovers are omitted. The numbers are very close except for the Eurocurrency maximum draw down. This can be explained due to the fact that the data before 1999 is different for the continuous data than it is for the individual contract. The continuous data prior to 1999 is the old euro currency (before it replaced the Deutsche mark) and the individual contract data is the old Deutsche mark multiplied by a certain constant. The smoothing process had very little impact on the actual ranges of the data and the relationship between the today's opening and closing prices. This leads us to believe that the difference occurs in the relationships of the closing prices over the past few days. The smoothing process keeps the current days open, high, low and close relationship intact, but seems to change the relationship of today's price action with prior days.

Miscellaneous Ramblings

Latest Version of the System Tester

I made just one modification to the System Tester software. I created a subroutine to check to see which order should be executed first. Right now the software would execute whichever order was placed first. It wasn't smart enough to check to see if a reversal order would occur prior to say a liquidation order. Just go to the website to download the latest version.

Download the System Tester code at
ftp://www.FuturesTruth.com/pub/SystemTester1.3.xls

If you have any questions on this article or code it contains, feel free to contact George directly at George@FuturesTruth.com.

Before you proceed make sure you have read the first part of the manual so that you will understand the evolution of the SystemTester software. In addition, you will gain much insight into how to properly use the software.

Think of testing a system as one big loop where we are looping through each day of historical data and taking action when our trade criteria is met.

Keep in mind that this is not a high priced finished product like TradersStudio or TradeStation. Expect to get your hands dirty playing around with the code. The development of this software was designed to be educational. With a little bit of elbow grease you can test and evaluate your own trading systems.

Here is a sample of a simple trading system. It buys on a stop at the Highest High of the past 20 days and sells (short) on the Lowest Low of the past 20 days. It incorporates $2000 protective stop and liquidates any loss after 10 trading days.

There are several built-in functions at your disposal. The following code provides the syntax for accessing them. Some are functions and some are subroutines. Make sure you use the right number of parameters for the functions/subroutines.

The most import subroutine is the Trade routine. This is the subroutine that you will use most often.

You call this subroutine whenever you ready to place an order.

Here are the different combinations one could use to place orders.

```
Call Trade(Buy, "MyBuyName" , Price , stp, i) - buys on stop
Call Trade(Buy, "MyBuyName" , myOpen, mkt, i) - buys market on Open
Call Trade(Buy, "MyBuyName" , myClose,mkt, i) - buys market on Close
Call Trade(Buy, "MyBuyName" , Price , lmt, i) - buys on limit
```

You can call the Trade subroutine with Sell, ExitLong, and ExitShort.

```
Call Trade(ExitLong, "MyExitLongName" ,Price, stp, i)
Call Trade(ExitShort, "MyExitShrtName" ,Price, stp, i)
Call Trade(Sell, "MySellName" ,Price, stp, i)
```

Also read the notes at the top of the actual VB code when you open the source code from the VB Editor. Also remember you must change the contract specifications in Module 2 inside the RunSystemTester() subroutine. You must know the PointValue, MinTick and the number of days you will need for your calculations. The SystemTester is preloaded with Crude Oil data. You can see how this subroutine is modified to work with this particular market. If any of the parameters are incorrect, you will have inaccurate results or Excel will halt execution. Copy and paste the data to test from the text files.

```
'****************************************************************
'****                 Main Trading Loop                      ****
'****************************************************************
Do While i <= numRecords

    i = i + 1                                         'Leave in.
    intraDayTrdCnt = 0                                'Leave in.

    If barsLong <> 0 Then barsLong = barsLong + 1      'Leave in.
    If barsShort <> 0 Then barsShort = barsShort + 1   'Leave in.
```

```
'****************************************************************
'**** This is a good place to put all of your function calls ****
'**** and system calculations.                               ****
'****************************************************************

    Call BollingerBand(myClose, 10, 2, avg, upBand, dnBand, i, 1)
    simpleAvg = Average(myClose, 10, i, 1)
    rsiVal = RSI(myClose, 14, i, 1)
    Call Stochastic(3, 4, 7, stoK, stoD, slowD, i, 1)

'****************************************************************
'****               Put All Of Your Orders Here              ****
'****************************************************************

    If marketPosition <> 1 Then
        Call Trade(Buy, "D-Buy", Highest(myHigh, 20, i, 1), stp, i)
    End If
    If marketPosition <> -1 Then
        Call Trade(Sell, "D-Sell", Lowest(myLow, 20, i, 1), stp, i)
    End If
    If marketPosition = -1 Then
        Call Trade(ExitShort, "ExitShortStop", entryPrice + 2000 / TickValue, stp, i)
        If barsShort > 10 And myClose(i) > entryPrice Then
            Call Trade(ExitShort, "10dayShOut", myClose(i), moc, i)
        End If
    End If
    If marketPosition = 1 Then
        Call Trade(ExitLong, "ExitLongStop", entryPrice - 2000 / TickValue, stp, i)
        If barsLong > 10 And myClose(i) < entryPrice Then
            Call Trade(ExitLong, "10dayLgOut", myClose(i), moc, i)
        End If
    End If

'****************************************************************
'****               End of Main Traiding Loop                ****
'****           No orders allowed below this point           ****
'****************************************************************
```