



MetaStock

Advanced Formula Writing

Formula Primer II

Printed in the USA
All Rights Reserved
Copyright © 2017

MetaStock
4548 South Atherton Dr, Suite 200
Salt Lake City, UT 84123
<http://www.metastock.com>

Table of Contents

Chapter 1: Introduction.....	5
An Approach to Formula Writing	5
The Proper Mind-set	5
What is not covered	6
 Chapter 2: A Firm Foundation	 7
Defining the end result	7
Know your environment	8
Know your tools	9
 Chapter 3: Customizing the Predefined.....	 10
Average True Range	10
A Wilder Smoothing Method	12
Volume with the RSI	13
 Chapter 4: Simulation Data	 17
What are Simulation Functions	17
Tracking Positions	17
Counting Equity	19
Trade Entry Price	21
That does not always work	22

Chapter 5: Line Studies 24

Horizontal lines	24
Fibonacci Retracement	25
Automatic Trendlines	29
Looking at this from a new angle	33
Channels	34

Chapter 6: The Logical Switch 38

Two state switches	38
Why uses switches	39
Three state switches	42
But wait there's more	43
Counting positions	44
There are limits	46

Chapter 7: Pivot Points..... 47

Middle of the road	47
Highs and Lows	50
Parts of a Whole	52
Building blocks	54

Chapter 8: Logical Twists..... 58

Dynamic constants	58
Normalization	59

Sequential Constructions	60
Divergences	62
Chapter 9: Custom Stops.....	66
Profit target	66
Trailing stop	68
Maximum Loss stops	69
Time based stops	70
Putting it all together	71
Chapter 10: Extrapolating Higher Time Frames.....	74
Starting Simple	74
Exponential Changes	77
Relative Applications	79
Stochastics	82
Chapter 11: Value to Generate Signal.....	87
A Simple Example	87
More Math	89
All is not so Simple	91
Chapter 12: Commentaries	92
What does a commentary do?	92
What is needed to write a commentary?	92

The Initial Framework	94
Adding Calculated Values	95
Conditional Logic	98
Coding the WriteIf()	99
Making the code shorter	103
Write the Exit States	103
The Last State	106
Final Touches	108
Appendix 1: Simulation Functions	109
Appendix 2: Glossary	112

Chapter 1: Introduction

An Approach to Formula Writing

Everybody is different and will approach problems and issues in different ways. Some people will look at the MetaStock Formula Language and may think of the various functions as tools in a workshop. Others may think of them as ingredients in kitchen. I have always loved puzzles and building things so for me, a more appropriate analogy would be Lego building blocks. With enough time, patience, and blocks, you can build just about anything.

The following sections will introduce the reader to concepts and ideas not covered in the Formula Primer. It will assume an understanding of the MetaStock Formula language as well as a strong proficiency in MetaStock. A good working knowledge of Algebra is also helpful. The reader may be at a disadvantage without these skills.

The new concepts will be presented as a task or problem to be solved. The issues that affect it will be discussed and then a solution explained. The solution presented may not be the best solution, but it will meet the requirements. In showing the process used to find the solutions, I hope to instill the reader with the knowledge and confidence to take what is presented and play. Rearrange the blocks, so to speak, and perhaps find something better.

If you are confident in your skills, you can skip to the last part of each solution. Should there be any questions on how that formula was created, please review the step by step process that preceded it. I have tried to be as complete as possible when explaining how each formula was created.

The Proper Mind-set

When I was just starting, I would be presented with an idea for a formula and my thoughts would always tend toward "That should be possible so how would I do it?" After making sure I understood all that a formula entailed, I would examine the specifications. A formula is almost always either a pattern to be recognized or a calculation to be performed. Occasionally, it will be both.

The first step is to break it apart, see what the different pieces are. If the formula requires three different conditions, I would treat each condition as a separate formula and write each part individually. If there was any doubt a part was not working correctly, I would test it by itself. Finally, I would combine them to make the finished formula.

The same approach applies when defining a pattern. Separate each part that makes up the pattern and write those parts. Then link them together to form the final formula. Taking my Lego analogy, if you want to build a house, you have to build 1) a base, 2) four walls, one of which must have a door, and 3) a roof. Build the parts and then combine them to form the house.

Sometimes parts will have to be further divided. Continuing the above example, one wall must have door. It may also need a window. Build the window and door, and then build the wall to contain them. Finally, add it to the model of the house.

Obviously, there are other approaches and some of them will work better, some will be worse. But don't let the size or complexity of the final product cause despair or thoughts of impossible solutions. Use what you know. If you need a shape (function) you don't have, think about how you might construct it with what you have. Not permitting something to be impossible is the first step to accomplishing it.

What is not covered

This text will not discuss external formula DLLs. There are several DLLs that have been created and circulated on the internet that do one or more of the tasks that are discussed in the following chapters. It is up to the author of any DLL to explain how to use his code. This text is focused on getting the most out of the MetaStock Formula Language.

It is beyond the scope of this work to explain how to create a DLL. The computer languages involved usually require a year or more of course work to learn. If such understanding is already possessed, the MetaStock Developer's Kit (MDK) provides all the further knowledge needed.

Chapter 2: A Firm Foundation

Defining the end result

"If you don't know exactly where you're going, how will you know when you get there?"

- Steve Maraboli, *Life, the Truth, and Being Free*

The question used for one's direction in life is equally valid in writing formulas. Before you can write a formula, you must know what you want it to do. Having a vague idea or a general concept is not enough. You can try many things to find a solution, but if you don't know what the solution looks like, how will you find it?

Consider the term "support". It has many meanings but for the moment, focus on the one pertaining to technical analysis. A rough definition is a price at which a security will stop downward movement and reverse direction. How is this identified? Support can be found at:

- Trendlines
- Channel bottoms
- Historical lows
- Indicators, like moving averages
- Fibonacci levels
- Pivot points

If a formula is to be written that uses support, a method of identifying that support must be defined. The method used will affect the final result as each method will likely find support at different price levels. In places they may be the same, or nearly so, but in others the values will be different, possibly considerably so.

There are similar issues with other common concepts. The general idea is understood, but the specifics are hazy. To write the formula, the method must be defined. If you do not know how something is defined or what something actually looks like, it will be very difficult to finish the formula.

An equally important question is how much. Some patterns, when they are expressed in English, will use terms like "close to", "large", and "long". While these terms, and others like them, may express and describe the concept, they are vague. Formulas do not do well with vague. If a formula is looking for something to be close to some value, the writer of the formula needs to define how close it has to be. Instead of saying something is large; define the size that makes it large. If

something has to exist for a long period of time, define how many bars are required to be a “long time.”

When defining terms, they may be set as a range of values. They may be assigned to variables so that part of the formula can be changed later. However, there must be some set value to work with.

When writing a computer program, this process is referred to as defining the program specifications. It is just as important when writing a formula. It is not possible to code "the closing price is close to the long-term moving average". You must specify the number of periods in the average. The formula can say the price is within some percentage of the value of the moving average or it's a certain distance or less from the average, but in both cases, there must be a specific, defined value.

Know your environment

The formula language was written to allow custom analytics in MetaStock. It is used to create indicators, explorations, system tests, and expert advisors. Those program features are the environment in which all formulas must operate. It is important to understand how those features work as their abilities and limitations affect what a formula can and cannot do.

To review, all formulas have certain restrictions:

- They can have not more than 20 variable definitions. All variables must start with a letter but may include numbers.
- They cannot have more than 20 numerical constants. These are individual numbers such as 2, 10, 20, etc.
- They have a limit to the number of characters allowed. This limit is higher in the indicator builder than in other tools, but the limit is still there

Indicator Builder

This is the only tool that can use input prompts in its formulas. A custom indicator can have up to six inputs. This allows the user of the indicator to vary its application without having to edit or change the actual formula.

Indicators are the only type of formula other formulas can reference. With the Fml() and FmlVar() functions, all or part of a custom indicator can be used in other formulas. If the custom indicator being referenced uses Input() functions, the default value of the inputs will be used for all references to the custom indicator.

Indicators can show several calculated values at once. A formula may include code for up to twenty different lines. Any reference to formulas that plot multiple lines by the Fml() function will only get the value of the last line.

The Explorer

Formulas for the Explorer cannot reference any value after the exploration date, even if the data would otherwise be available.

The Explorer allows formulas to be placed in different columns. The last value of those columns can be accessed by the filter formula with ColA, ColB, etc commands. The column references do not have any historical values and will be undefined if the filter tries to access earlier values.

The System Tester

Optimization is only possible in the system tester. The OPT variables are not defined anywhere else. If a formula with OPT variables is copied and pasted to another formula based tool, the OPT variables will have to be changed to fixed constants.

Simulation functions are also unique to the system tester. These functions let you access the current value of different aspects of a system test. They can only give data for the current bar. The simulation functions will be undefined if used in any reference statement, like Ref(), Value-When(). They can not be used in any function that uses historical or future data.

Expert advisor

This is a catch-all tool. It can color the price bars; place symbols or a ribbon on a chart; and generate alerts that pop up to advise users a condition has been met. It also has a commentary window. The commentary uses WriteIf() to define blocks of text to be displayed based on the conditions expressed by formulas. WriteVal allows the text to display different values based on the calculations of a formula. Commentaries are not difficult but due to their length, may be more challenging.

Know your tools

The functions of the formula language are the tools used to construct formulas. Some are general math and logic functions that will be used in almost every formula. Many are for specific indicators and will be used only when that indicator's value is needed. Of the indicator functions, some are so useful, you will find them to be stalwarts of your formula writing.

Each function is described in detail in the Help section of MetaStock. They are further elaborated on in the formula primer. This text will not go over those again. Familiarity with these functions is critically important. Sometimes it is possible to replace one function with several others, but not always. Just like it is very hard to cut a piece of wood to the right size if you do not have a saw, so it is difficult to write some formulas without specific functions. It cannot be stressed enough; knowing what the different functions do will help any aspiring formula writer.

Chapter 3: Customizing the Predefined

MetaStock comes with over one-hundred built-in indicators. Creating custom formulas that duplicate the calculations of built-in indicators helps build familiarity with the formula functions. The process of reading how the formula is calculated and then expressing that in the MetaStock Formula Language builds confidence and offers opportunities for insight and discovery. Such custom versions of indicators will be needed to proceed with creating the formulas requested in this chapter's exercises.

The formula language allows new indicators to be created. However, what happens if a customized indicator is required? This is not just a different number of periods, but a difference in how the calculation is performed. It is not possible to alter the MACD function and have it use simple moving averages instead of exponential ones. There is no way to change the smoothing method of the stochastic oscillator or the RSI.

The only solution is to build a new custom indicator. The new indicator will have to exactly match the original indicator. Then the change can be made to create the customized calculation.

If you are good in math and have the exact formula for the original indicator, it is possible to skip straight to the custom formula. After all, why bother making a window if you plan to remove the glass to make a door? However, sometimes the math can be misleading.

Average True Range

MetaStock calculates the Average True Range (ATR) as it was defined in Wells Wilder's book. This requires using a custom averaging method invented by Mr. Wilder. However, other programs use the standard exponential moving average instead. This will yield different results.

Articles published in Technical Analysis of Stocks and Commodities magazine (TASC) have taken sides on this. One mathematician presented a proof that the two averaging methods were identical. Yet the ATR in MetaStock would give a different result than the ATR in other programs.

The answer to this conundrum is that Mr. Wilder's smoothing method becomes equal to the exponential moving average. Using 14 periods for the smoothing time period, an exponential moving average will initially yield a different result than Wilder's smoothing method. If the calculation is extended over several thousand bars of data, the values will merge and become indistinguishable.

Exercise 1

Suppose the exponential smoothing version of the ATR is desired but the chart only contains a hundred bars (about 5 months of daily data). This is not enough time for the two smoothing methods to merge. Write a custom version of the ATR calculated with a 14 period exponential moving average.

Solution

Step 1: An Analysis

This is straight forward. Take the traditional ATR and change it to use an EMA instead of Wilder's Smoothing. Since the ATR is a common indicator, its calculation is easily found. The help in Metastock lists it this way:

The True Range indicator is defined by Wilder to be the greatest of the following for each period:

The distance from today's high to today's low.
The distance from yesterday's close to today's high.
The distance from yesterday's close to today's low.

The Average True Range is simply the average of the true ranges over some number of periods specified by the user.

Mr. Wilder's smoothing method is to use a simple moving average for the first calculation. For each subsequent bar, he does the following:

1. Start with the value the average had on the previous bar
2. Subtract the value of the average divided by the time periods
3. Divide the current value of the data being averaged by the time periods
4. Add the value from step 3 to the result from step 2.

This gives the new value of the average.

Step 2: The Logic

This calculation could be expressed this way:

Truerange = largest value of:

- today's high minus today's low
- today's high minus yesterday's close
- yesterday's close minus today's low

The true range can be expressed in MetaStock with the Max() function:

```
Truerange:= Max( H - L, Max( H - Ref( C, -1), Ref( C, -1) - L) );
```

However, using algebra, it can be simplified to this:

```
Truerange:= max( H, Ref( C, -1)) - Min( Ref( C, -1), L);
```

The solution could jump straight to the EMA but it is good practice to go through the full process. The ATR is the average of the true range. Fortunately, MetaStock includes a function to calculate the Wilder smoothing method, Wilders().

```
Truerange:= max( H, Ref( C, -1)) - min( Ref( C, -1), L);  
Wilders( Truerange, 14)
```

Step 3: The Final Formula

Changing the formula to use an exponential moving average instead of Wilder's smoothing method will result in the final formula:

```
Truerange:= max( H, Ref( C, -1)) - min( Ref( C, -1), L);  
Mov( Truerange, 14, E)
```

A Wilder Smoothing Method

The last example did not truly calculate the formula down to the basics. The Wilder's Smoothing was done with the Wilder's function. This was fine for the requirements given, but as will be shown later, there are times the indicator must be calculated without any pre-defined functions

Exercise 2

Write an indicator that calculates a 14-period Wilder's Smoothing of the Close. Do not use Wilders() or any moving average functions

Solution

Step 1: An Analysis

The calculation of Wilder's Smoothing was listed before but here it is again for convenience:

Use a simple average for the first calculation. For each subsequent bar, do the following:

1. Start with the value the average had on the previous bar
2. Subtract the value of the average divided by the time periods
3. Divide the current value of the data being averaged by the time periods
4. Add the value from step 3 to the result from step 2.

Step 2: The Logic

An If() will have to be used to start the calculation with the simple moving average. It will need to be TRUE only on the first bar the Wilder's smoothing is calculated on. Since this is a 14-period average, that will be the 14th bar of data. The Cum() function applied to the number 1 will count the bars. When this has a value of 14, the first 14 close prices will need to be totaled and then divided by 14. Subsequent bars will employ the PREV function to access the previous value of the average. The logic could be written this way:

```
If bar number equals 14,  
Wilder's Smoothing equals a simple average of the last 14 closing  
prices  
else  
Wilder's Smoothing equals previous value minus previous value  
divided by 14 plus current Close divided by 14
```

The first part of the If() with the The Cum() function could be written this way:

```
If ( Cum ( 1 ) = 14 ,
```

When the conditional expression is True, a summation of the closing prices will allow the calculation of the simple average:

```
Sum ( C , 14 ) / 14 ,
```

This has given two of the three parts to the formula. Last part requires the use of PREV:

```
PREV + ( C / 14 ) - ( PREV / 14 )
```

This last part can be shortened by combining C and PREV before dividing by 14:

```
PREV + ( ( C - PREV ) / 14 )
```

Step 3: The Final Formula

Putting all the parts together, the final formula for Wilder's Smoothing is:

```
If ( Cum ( 1 ) = 14 , Sum ( C , 14 ) / 14 ,  
PREV + ( ( C - PREV ) / 14 ) )
```

Volume with the RSI

Now it is time to apply this in a larger formula. The Relative Strength Index (RSI), also by Wells Wilder is very well known. It has even been modified a few times by others seeking to embellish on it. The basic calculation is:

1. calculate the point change in the Close price from one bar to the next.
2. using Wilder's smoothing, average all the positive moves over 14 periods
3. use Wilder's smoothing, average all the negative moves over 14 periods. Use the absolute value to keep this a positive number
4. divide the positive move average by the negative move average
5. divide 100 by sum of 1 plus the ratio of the positive and negative moves
6. subtract the last calculation from 100

This can be expressed in a MetaStock formula this way:

```
change:= ROC(C,1,$);
Z:=Wilders(If(change>0,change,0),14);
Y:=Wilders(If(change<0,Abs(change),0),14);
RS:=Z/Y;
100-(100/(1+RS))
```

Suppose a trader wanted have the RSI reflect volume as well as price. Change the calculation to include volume.

Exercise 3

Determine if volume is increasing or decreasing. Change the calculation of the wilders smoothing so the new value is subtracted on decreasing volume and added on increasing volume. The values of the upmove (Z) and downmove (Y) should not go below zero. Add code to prevent divide by zero errors. If the down move average is equal to zero, the RS ratio should check the value of the up move average. If the up move average is also zero, the ratio should equal 50. otherwise, it should be 100.

Solution

Step 1: An Analysis

The calculation of the Wilders average will need to be done by custom calculation instead of the Wilders() function. The divide by zero error trap will also require an additional variable and some If() conditions. To have the modified RSI equal 50 when both averages are zero, the ratio will need to equal 1. The math does not allow for the RSI to reach 100, but it can get close if the denominator is made a really small number.

Step 2: The Logic

The first line of the RSI does not need to be changed. It is just a shorter way to write the rate of change calculation. It is assigned to a variable for easy reference in the up and down move averages.

```
change:= ROC(C,1,$);
```

The check of the volume increasing or decreasing can be added next. Since this will determine if the change is added or subtracted from the previous average, this will be written as a multiplier and will have the value of 1, -1, or zero if the volume is the same as the previous bar:

```
vmod:= If (V>Ref (V, -1) , 1, If (V<Ref (V, -1) , -1, 0) ) ;
```

The Z and Y variables of the original RSI need to be expanded to not use the Wilders smoothing. Since the If() to check whether change is up or down is rather long, it will be set as a variable and then the average will be done as a separate variable. Compare these formulas to the Wilder's Smoothing from the previous exercise:

```
Z1:=If (change>0, change, 0) ;
Z:=If (Cum(1)= 14, Cum(Z1)/14, PREV + ((Z1 - PREV)/14)) ;
Y1:=If (change<0, Abs (change) , 0) ;
Y:=If (Cum(1)= 14, Cum(Y1)/14, PREV + ((Y1 - PREV)/14)) ;
```

Y and Z still need the volume change. In order to apply the volume modifier just to the value of the current bar, the Wilder's Smoothing will be expanded to divide the new change value and the previous average values separately. Then since these averages can not go negative, the second half of the If() logic will be put inside a Max() function:

```
Z:=If ( Cum(1) = 14, Mov (z1,14,S) ,
Max (PREV - (PREV/14) + (Z1/14)*vmod, 0) ) ;
Y:=If ( Cum(1) = 14, Mov (Y1,14,S) ,
Max (PREV - (PREV/14) + (Y1/14)*vmod, 0) ) ;
```

All that remains is the code to prevent divide by zero. This can be done through a couple of If()s. First the denominator is assigned to a variable. The new variable sets a flag if the value would be zero but otherwise has the value of Y.:

```
denom:= If (Y=0, -1, Y) ;
```

Now RS checks the value of denom. If the flag shows Y was zero, it either assigns RS to 1 or divides Z by 0.0001. If Y was not equal to zero, it divides Z by the value of Y (stored in denom).

```
RS:=If (denom=-1, If (z=0, 1, Z/0.0001) , Z/denom) ;
```

By setting the value to 1, the next step of the calculation, dividing 100 by the sum of 1 plus the RS ration will result in the value of 50. When that is subtracted from 100 in the final step, the modified RSI will report a value of 50.

Step 3: The Final Formula

putting all this together, the final formula is:

```
change:= ROC (C,1,$) ;
```

```

vmod:= If (V>Ref (V,-1), 1, If (V<Ref (V,-1), -1, 0));
Z1:=If (change>0, change, 0);
Z:=If ( Cum(1) = 14, Mov (z1,14,S),
  Max (PREV - (PREV/14) + (Z1/14)*vmod, 0) );
Y1:=If (change<0, Abs (change), 0);
Y:=If ( Cum(1) = 14, Mov (Y1,14,S),
  Max (PREV - (PREV/14) + (Y1/14)*vmod, 0) );
denom:= If (Y=0, -1, Y);
RS:=If (denom=-1, If (z=0, 1, Z/0.0001), Z/denom);
100 - (100 / (1+RS))

```

This text does not presume to say this formula is better than the original RSI. This was merely an exercise on how to modify a built-in formula.

Chapter 4: Simulation Data

What are Simulation Functions

The simulation functions were not discussed in the Formula Primer. They were added to MetaStock with the Enhanced System Tester in version 8.0. Below is a synopsis of how these functions are different from the other formula functions.

- They can only be used in the system tester. If used anywhere else, they will not give any syntax errors, but the formula will be undefined and return no value
- They are calls to a DLL. However, unlike other DLL calls, these do not require the use of the ExtFml() function.
- Use of these functions will cause the system tester to calculate more slowly
- They have no value in a formula except for the current bar. This means they will be undefined if nested inside almost any other function. They cannot be averaged over time. They do not have highest high values or lowest low values. They cannot be referenced or used with BarsSince() or ValueWhen().

A complete list of the simulation functions is in the appendix (see page 109). It is worthwhile to familiarize yourself with them. While these functions are limited there are certain things that cannot be done without them. For instance, the system tester allows simulations to have multiple positions open at once. Consider a system that wants to allow two long positions but under different conditions:

Tracking Positions

Exercise 1

Write a system test for the following trading system:

Initial trade - this position will be opened when the close crosses above a 20-period simple moving average of the close

Second position - will be opened after the close has been above the 20-period simple moving average for at least five consecutive bars and both the close and the moving average are increasing.

Exits - all open positions will be closed when the close falls below the 20-period moving average.

Solution

This could be coded without the simulation functions but it is much easier with them.

Step 1: An Analysis

This system test has three signals, an exit signal and two entry signals. One entry signal is a single condition but the other has four conditions that must all be true.

Step 2: The Logic

When multiple conditions are involved, it sometimes helps to layout the logic and then replace the conditions with the correct code on a step by step basis.

Enter Long Signal:

```
( close crosses MA )
OR
(
another long trade open
AND
Close greater than MA for 5 consecutive bars
AND
Close has increased from previous bar
AND
The MA has increased from the previous bar
)
```

Exit Long signal

Close falls below the MA

All of these conditions except the requirement for an existing trade should be easily done from the information in the Formula Primer. The only one that might cause any trouble is the requirement of the close being above the average for five consecutive days. This one is met if the trick with the Sum() function is used. Putting in the code for all but the number of current trades, the formulas are:

Enter Long Signal:

```
cross( C, Mov( C, 20, S)) OR
( another long trade open AND
Sum( C > Mov( C, 20, S), 5)=5 AND
Roc( C, 1, $)>0 AND
Roc( Mov( C, 20, S), 1, $)>0)
```

Exit Long signal

Cross(Mov(C, 20, S), C)

A look through the list of simulations functions shows two that would work:

```
Simulation.LongPositionCount  
Simulation.PositionCount
```

Since the system may be expanded to in the future to include short trades, the one that specifies long positions only should be used

```
(Simulation.LongPositionCount = 1
```

Step 3: The Final Formula

Adding that function to the previous code gives the final system listed below:

Enter Long Signal:

```
cross( C, Mov( C, 20, S)) OR  
(Simulation.LongPositionCount = 1 AND  
Sum( C > Mov( C, 20, S), 5)=5 AND  
Roc( C, 1, $)>0 AND  
Roc( Mov( C, 20, S), 1, $)>0)
```

Exit Long signal

```
Cross(Mov(C, 20, S), C)
```

Counting Equity

The Enhanced System Tester includes the ability to calculate the size of a trade with a formula. Many published systems use a variable trade size as part of money management. The simulation functions allow MetaStock to implement such techniques.

Exercise 2

A trading system uses the opening price to verify a signal generated on the previous bar. The system will enter at the open plus \$0.25 if the signal is verified. The size of the trade is based on a risk calculation. A stop will be set at a 40-period simple moving average minus twice a 14-period ATR.

The risk is defined as the difference between the entry price and the value of the ATR stop on the signal bar (not the entry bar). To get the trade size, calculate the risk as a percentage of the entry price. Multiply that percentage by three to get the total risk percentage. Then multiply that by the current equity to get the trade size in dollars.

Solution

Step 1: An Analysis

While the description is long, the calculation is fairly straight forward. Risk is can be written as:

```
(Open + 0.25) - Ref (Mov (c, 40, s) - ATR (14) , -1)
```

Likewise, the trade size can be expressed by:

```
( 3 * ( risk / ( Open + 0.25 ) ) ) * total equity
```

Step 2: The Logic

Start by defining what is known. Assign risk to a variable so it can be altered later if a different calculation is desired. This also helps to keep the formula easier to read.

```
Risk:= (Open + 0.25) - Ref (Mov (c, 40, s) - ATR (14) , -1) ;
```

The percentage of equity to use for the trade size can now be defined as:

```
Tradesize:= ( 3 * ( risk / ( Open + 0.25 ) ) ) ;
```

The rest of the formula for the trade size can be expressed as:

```
Equity:= simulation functions ;  
Equity * tradesize
```

There is not a single function that will give the total equity. The system test divides the equity among several "accounts" to simulate the margin requirements and variations of open trades. The different accounts are:

Account Name	Purpose
Cash	contains all equity not allocated to an open position
Borrowed	money owed the broker for losing leveraged trades (entered on a margin)
Reserved	money set aside to cover open leveraged trades
Portfolio Value	The value of all open positions if they were to be closed on the current bar.

Each account has a simulation function to get the amount of equity it currently contains. These separate functions have to be added together to get the total equity. After doing so, the formula variable for the equity will look like this:

```
Equity:= Simulation.AccountCash + Simulation.AccountBorrowed +  
Simulation.PortfolioValue + Simulation.AccountReserved;
```

Step 3: The Final Formula

With the equity defined, the final formula can be written:

```
Risk:= (Open + 0.25)-Ref(Mov(c,40,s)-ATR(14), -1);  
Tradesize:= ( 3 * ( risk / ( Open + 0.25 ) ) );  
Equity:= Simulation.AccountCash + Simulation.AccountBorrowed +  
Simulation.PortfolioValue + Simulation.AccountReserved;  
Equity * tradesize
```

Trade Entry Price

Many times a system will specify the entry price of a trade. This could be used for a variety of stops or exit conditions. However, there is no simulation function for the entry price. This does not mean the price cannot be derived.

Exercise 3

A system requires all trades be exited after the price has risen some percentage from the entry price. End of day versions of MetaStock use to not allow optimizing stops. If the percentage is to be optimized, it has to be included in the formula. Use OPT1 for the percent and write a formula to exit when this profit target has been reached.

Solution

Step 1: An Analysis

This system wants to exit after a minimum profit is reached. That means the High must exceed the profit level. The only real issue is finding the entry price.

Step 2: The Logic

The formula is fairly straight forward:

```
H > (entry price + optimized percentage)
```

The optimized percentage will be OPT1 divided by 100 and then added to 1. This will give a decimal value which can be multiplied times the entry price to get the target value:

```
Optpercent:= (OPT1 / 100) + 1;
```

A search through the simulations functions will yield this function:

```
Simulation.CurrentPositionPointDifference
```

In a long position, the current closing price minus that function should give the price the trade was entered at.

```
C - Simulation.CurrentPositionPointDifference;
```

Note: This may not always work. See the information after the solution for more details.

Step 3: The Final Formula

With the simulation function, the entry price can be calculated and this formula written:

```
Optpercent:= (OPT1 / 100) + 1;  
Entryprice:= C-Simulation.CurrentPositionPointDifference;  
H > entryprice * optpercent
```

If this was for a short position, the Simulation.CurrentPositionPointDifference function would be added to the close of the current bar.

```
C + Simulation.CurrentPositionPointDifference
```

That does not always work

As noted above, the Simulation.CurrentPositionPointDifference function does not resolve all entry price issues. The function calculates the profit of the position. To do that, it has to calculate the difference between the entry price and a current price. However, since the System Tester has no way to know the price the trade will exit on, it has to make an assumption.

The Trade Options of the System Tester has a tab with Trade Execution settings. There, a default price field can be set which long and short trades will use for entering and exiting trades. The entry price will not affect the simulation function but the long exit and short exit values will. Whatever price field is set there has to be used in the entry price formula or the calculation will be inaccurate.

To elaborate, the above solution subtracted the Simulation.CurrentPositionPointDifference function from the closing price. However, if the trade execution settings had the long exit price field as the Open, the formula would be wrong. The Simulation.CurrentPositionPointDifference function would report the difference between the entry price and the open of the current bar. When that value is subtracted from the close, the formula would be off by the difference between the current bar's open and closing prices.

There is no way to query or determine what price field the trade execution settings will use. The formula can include a comment specifying what is expected, but this will not prevent inaccurate results. Something more is required. This will be further explored and developed in Chapter 9

Chapter 5: Line Studies

MetaStock has many indicators and the formula language allows others to be created. However, MetaStock also has a good selection of line studies. These are drawing tools used to annotate a chart. As good as these tools are, MetaStock does not include a way to automate them. The formula language does not have the ability to do macros nor can it create new drawing tools. However, the formula language can make indicators that mimic some line studies.

Horizontal lines

Most lines studies are just straight lines. Any line can be defined with the formula:

$$F(n) = b + (y * x)$$

Where:

$F(n)$ is the value of the line on any given bar where n is the bar number

b is a constant value

y is the slope of the line

x is the number of bars since the line started. It will always be equal to n minus 1

Since x is equal to n minus 1, on the first bar, x will be equal to zero. Therefore, on the first bar of the chart the formula would be equal to:

$$F(1) = b + (y * 0)$$

Or

$$F(1) = b$$

This means b is the starting point of the line.

The slope is how much a line will increase or decrease as it moves from one bar to the next. When the line is flat, the slope is zero. That formula would be:

$$F(n) = b + (0 * x)$$

Or

$$F(n) = b$$

Therefore the horizontal line is easy to code. A flat line is just a constant. The line study can be simulated with a formula that prompts for a value and plots it on every bar in the chart.

Exercise 1

Create a formula that draws a horizontal line. It should prompt for the value to use. The prompt should accept any number between -1000 and +1000.

Solution

Step 1: An Analysis

The input function is used for the prompt. This was covered in the Formula Primer. Please refer there or to the MetaStock help for information on its use.

The exercise did not specify what the default should be for the input. Since the line allows both positive and negative values, zero seems a logical middle point for the default.

Step 2: The Logic

Using the formula above, for a horizontal line, the value on any bar is the constant b. b, in this case, is the value from the input function

```
B:= Input("value of the line",-1000,+1000,0);
```

Step 3: The Final Formula

To finish this formula, add a second line that just call the variable B:

```
B:= Input("value of the line",-1000,+1000,0);  
B
```

Fibonacci Retracement

A Fibonacci Retracement is drawn as a set of horizontal lines similar to the previous formula. The difference is in calculating where those lines should be drawn. They are based on price move. The distance of that move is calculated and then multiplied by set percentages. The resulting values are then adding to the end point of a down move or subtracted from the end of an up move.

The process sounds simple but it does not explain how to measure the move? The formula needs:

1. a point in time to start drawing the lines from
2. a base value to add or subtract the fibonacci values from
3. the distance of the move to calculate the fibonacci values
4. whether this is an up move or a down move

Exercise 2

Write a formula to calculate and draw Fibonacci Retracement lines on a chart. Prompt for any needed values. The formula should be able to draw retracements on both up and down moves. Assume the formula will only be used on daily or longer time frames. The formula should draw lines at the percentages of 0.0, 23.6, 38.2, 50, 61.8, and 100.0

Solution

Step 1: An Analysis

A formula can only have six inputs. There are four values needed. One of the values is the end point of the move. If the date of the end of the move is known, ValueWhen() can be used to get the base value. So now only three values are needed:

1. the end point
2. the move distance
3. the move direction

The formula could prompt for those values. Alternately, the formula can prompt for the starting point of the move and then calculate the other two values. This formula will use the second option.

Step 2: The Logic

To get a date, the formula needs the day of the month, the month, and the year. Three values each for two dates is a total of six inputs. So all the allowed inputs will be used. They should look something like this:

```
sm:=Input("start month",1,12,1);
sd:=Input("start day",1,31,1);
sy:=Input("start year",1970,2460,2014);
em:=Input("end month",1,12,1);
ed:=Input("end day",1,31,1);
ey:=Input("end year",1970,2460,2014);
```

With these values, a conditional statement can be built that will be true only on a specific date. It will compare the inputted values with the DayOfMonth(), Month() and Year() functions of MetaStock. One will be constructed for both the beginning and ending date:

```
sdt:= Month()=sm AND DayOfMonth()=sd AND Year()=sy;
edt:= Month()=em AND DayOfMonth()=ed AND Year()=ey;
```

Now, ValueWhen() can be used to get the close on each date.

```
svalue:= ValueWhen(1, sdt, C);  
evalue:= ValueWhen(1, edt, C);
```

By taking the difference between the two prices, the distance of the move is found, as is the direction. The absolute value of the distance is used for the percentage calculations. The direction is set as a variable to which is assigned a +1 for a down move and -1 for an up move. This may seem incorrect but it makes the logic easier later on.

```
dis:= Abs(evalue-svalue);  
dir:= If(evalue < svalue, 1, -1);
```

Now the actual lines can be drawn. The close of the end date is used as the base line. To that is added the move distance times the move direction times the individual percentages. Since the dir variable uses -1 for up moves, it will subtract the distance for the retracements. The +1 will add the distance after down moves. For the 0 line, nothing needs to be added to the base. For the 100 line is just multiplied by the dir variable.

```
evalue;  
evalue + (dis * dir * 0.236);  
evalue + (dis * dir * 0.382);  
evalue + (dis * dir * 0.5);  
evalue + (dis * dir * 0.618);  
evalue + (dis * dir)
```


Step 3: The Final Formula

Putting all this together, the final formula is:

```
sm:=Input("start month",1,12,1);
sd:=Input("start day",1,31, 1);
sy:=Input("start year",1970,2460,2014);
em:=Input("end month",1,12,1);
ed:=Input("end day",1,31, 1);
ey:=Input("end year",1970,2460,2014);
sdt:= Month()=sm AND DayOfMonth()=sd AND Year()=sy;
edt:= Month()=em AND DayOfMonth()=ed AND Year()=ey;
svalue:= ValueWhen(1, sdt, C);
evalue:= ValueWhen(1, edt, C);
dis:= Abs(evalue-svalue);
dir:= If(evalue < svalue, 1, -1);
evalue;
evalue + (dis * dir * 0.236);
evalue + (dis * dir * 0.382);
evalue + (dis * dir * 0.5);
evalue + (dis * dir * 0.618);
evalue + (dis * dir)
```

Below is an example of how this would look in a chart. A trendline was added to show the move represented by the dates entered.



Note: If a date is entered that does not exist in the chart, the formula will fail. ValueWhen() looks for the exact condition of a bar having the day, month, and year specified. If such a bar does not exist in the available data, ValueWhen() will be undefined. This will cause the formula to plot nothing.

Automatic Trendlines

The same prompts that retrieved the dates for the Fibonacci Retracement can be used to draw a trendline. Instead of calculating the distance of the move, the formula needs to calculate the slope.

Exercise 3

Create a formula to draw a trendline from two dates supplied by prompts. Modify the previous prompts to allow for intraday use. The formula should allow for hour and minute values, but not require them.

Solution

Step 1: An Analysis

The prompts are the most immediate issue here. The formula needs to get five values for two different dates, a total of ten values. The formula language does not allow this so the values have to be combined. Hour and minute values can be combined into a single number using 24-hour notation. A similar combination will allow the month and day of month to be combined. By prompting this way, the formula is still requesting just six inputs but it is actually receiving ten values.

Step 2: The Logic

The formula is also suppose to not require the intraday values. If a custom indicator plots the value of hour() on a daily chart, the result will be a flat line with the value of zero. A formula plotting the value of the Minute() function will have the same result. Daily charts have a time stamp of 0:00. Therefore, if the time prompt has this value as the default, then the time can be ignored or entered as the user desires.

The modified prompts for date and time values are just a minor change from the previous formula. The expected format of the input is included in brackets.

```
sdm:=Input("start day and month [ mmdd ]",1,1235,103);
sy:=Input("start year",1970,2460,2014);
st:=Input("start time [ 24 hour - hhmm ]",0,2360,0);
edm:=Input("ending day and month [ mmdd ]",1,1235,103);
ey:=Input("start year",1970,2460,2014);
et:=Input("ending time [ 24 hour - hhmm ]",0,2360,0);
```

Now the formula needs to extract the values from the prompts. This will require parsing the inputted values. The hour can be extracted from the time prompt by dividing time by one hundred and applying the Int() function to remove the fractional amount.

```
(Hour() = Int(st/100))
```

The minute value can be obtained in almost the same way. Divide the time by one hundred. Now apply the Frac() function to remove all but the fractional value. Multiply the result by one hundred and apply the Rnd() function just in case the math somehow generated a rounding error.

```
(Minute() = Rnd(Frac(st/100)*100));
```

This same logic can be used for the month and day values. Doing all the parsing for both dates will give the following logic for the starting and ending points:

```
sdt := (Month() = Int(sdm/100)) AND  
(DayOfMonth() = Rnd(Frac(sdm/100)*100)) AND (Year() = sy) AND  
(Hour() = Int(st/100)) AND  
(Minute() = Rnd(Frac(st/100)*100));  
edt := (Month() = Int(edm/100)) AND  
(DayOfMonth() = Rnd(Frac(edm/100)*100)) AND (Year() = ey) AND  
(Hour() = Int(et/100)) AND  
(Minute() = Rnd(Frac(et/100)*100));
```

Once again, ValueWhen() will get the value of the close for the dates.

```
svalue := ValueWhen(1, sdt, C);  
evalue := ValueWhen(1, edt, C);
```

Now, the formula also needs a way to measure the distance, or number of bars, between those values. There is not a built-in function for the bar number, but there is an easy way to create one. Cum() adds a value to a running total starting from the first bar loaded. If the value to add is the constant 1, then Cum() is effectively counting the number of bars in the chart. Use Cum(1) inside the ValueWhen() function to get the horizontal position in the chart of the two points.

```
sbar := ValueWhen(1, sdt, Cum(1));  
ebar := ValueWhen(1, edt, Cum(1));
```

There is a problem with the values calculated for the start and end points. ValueWhen() is only defined after the condition has occurred. The start value function is defined once the start date is reached. However, the end date must be reached before those values are defined. to calculate the slope and therefore draw the line, both values must be defined at the beginning of the line. By wrapping the values inside the LastValue() function they will be available anywhere in the chart.

```
svalue:= LastValue(ValueWhen(1, sdt, C));
evaluate:= LastValue(ValueWhen(1, edt, C));
sbar:= LastValue(ValueWhen(1, sdt, Cum(1)));
ebar:= LastValue(ValueWhen(1, edt, Cum(1)));
```

The slope of a line is generally defined as Rise over Run. That is, the change in the y-axis values divided by the change in the x-axis. Using the four values for price and bar number, the slope can be written this way:

```
slope:= (evaluate - svalue) / (ebar - sbar);
```

There is now enough information to calculate the line. Once again, the formula for a line is:

$$F(n) = b + (y * x)$$

B is the closing price on the start date. Y is the slope that was just defined. All that remains is x. X is the number of bars since the start date. The BarsSince() function can supply this.

```
svalue+(BarsSince(sdt)*slope)
```

Step 3: The Final Formula

Putting all this together, the final formula would be:

```
sdm:=Input("start day and month [ mmdd ]",1,1235,103);
sy:=Input("start year",1970,2460,2008);
st:=Input("start time [ 24 hour - hhmm ]",0,2360,0);
edm:=Input("ending day and month [ mmdd ]",1,1235,103);
ey:=Input("start year",1970,2460,2008);
et:=Input("ending time [ 24 hour - hhmm ]",0,2360,0);
sdt:=(Month()=Int(sdm/100)) AND
(DayOfMonth()=Rnd(Frac(sdm/100)*100)) AND
(Year()=sy) AND
(Hour()= Int(st/100)) AND
(Minute()=Rnd(Frac(st/100)*100));
edt:=(Month()=Int(edm/100)) AND
(DayOfMonth()=Rnd(Frac(edm/100)*100)) AND
(Year()=ey) AND
(Hour()= Int(et/100)) AND
```

```

(Minute()=Rnd(Frac(et/100)*100));
svalue:= LastValue(ValueWhen(1, sdt, C));
evalue:= LastValue(ValueWhen(1, edt, C));
sbar:= LastValue(ValueWhen(1, sdt, Cum(1)));
ebar:= LastValue(ValueWhen(1, edt, Cum(1)));
slope:= (evalue - svalue) / (ebar - sbar);
svalue +(BarsSince(sdt)*slope)

```

Below is an example of how this formula would on a chart:



Notice the line starts at the start date. This is because the BarsSince() function is undefined until the specified condition occurs. Since the condition is a unique date, that date must occur for the formula to start. This is similar to the way the Fibonacci lines did not start until the end date because ValueWhen() was undefined until that date.

There was an alternative to parsing the imputed dates. The formula could have combined the values the input were compared to. Doing it that way, the checks for the dates would have been written this way:

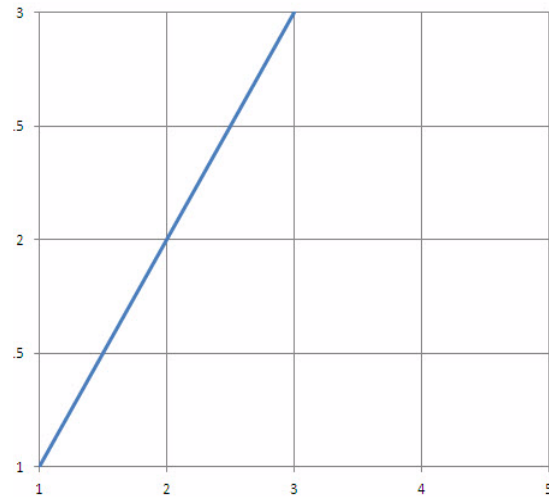
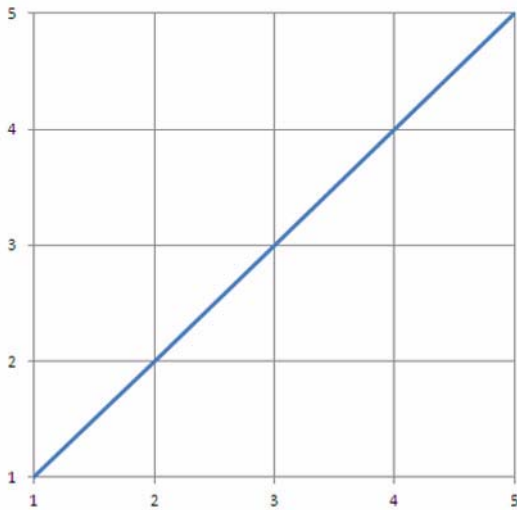
```

sdt:= (sdm = Month()*100 + DayOfMonth()) AND
(sy = Year()) AND (st = Hour()*100 + Minute());
edt:= (edm = Month()*100 + DayOfMonth()) AND
(ey = Year()) AND (et = Hour()*100 + Minute());

```

Looking at this from a new angle

Sometimes it is desirable to draw a line at a certain angle. This is problematic since a mathematical angle is usually not the same as the visual angle. Consider the following two graphs. They both show a mathematical forty-five degree angle:



Mathematically, forty-five degrees means that for every one point increase in the y axis, the x axis also increases one point. Visually a forty-five degree angle is the bisecting line that divides a square into two triangles. Of the two, the mathematical angle is easier to code.

Exercise 4

Write a formula that prompts for a date, a starting point, and an angle. The formula will then draw a line from that point at the specified angle. Limit the allowed angle to between +60 degrees and -60 degrees.

Solution

Step 1: An Analysis

Prompting for the date and starting point should be routine by now. The trick to this formula is converting the angle to a slope. Geometrically the slope of an angle is equal to the Sine of the angle divided by the Cosine of the angle.

Step 2: The Logic

Here are the prompts and the date parsing. To keep it simple the date does not include time. Prompts were also added for the starting value and the angle of the line

```
sdm:=Input("start day and month [ mmdd ]",1,1235,103);
sy:=Input("start year",1970, 2360,2008);
svalue:=Input("start value", 0, 30000, 100);
an:=Input("angle of the trendline",-60, 60, 45);
sdt:= (sdm = Month()*100 + DayOfMonth()) AND (sy = Year());
```

The Sine and Cosine functions are Sin() and Cos(). The slope would then be calculated this way:

```
slope:= Sin(an)/Cos(an);
```

From here the code is the same as the previous trendline formula

```
svalue +(BarsSince(sdt)*slope)
```

Step 3: The Final Formula

Combining the code above will give the following formula:

```
sdm:=Input("start day and month [ mmdd ]",1,1235,103);
sy:=Input("start year",1970,2360,2008);
svalue:=Input("start value", 0,30000, 100);
an:=Input("angle of the trendline",-60,60,45);
sdt:= (sdm = Month()*100 + DayOfMonth()) AND (sy = Year());
slope:= Sin(an)/Cos(an);
svalue +(BarsSince(sdt)*slope)
```

But what happens if the visual angle is desired? Here, the formula can not be perfect as it must rest on assumptions. Theoretically, the formula could use the normalization method (discussed in Chapter 8 on page 59). This would require the formula to prompt for the number of bars being displayed, calculate the ratio of the scale height in points versus the number of bars and use that ratio to adjust the mathematical slope. However, there are problems with this. It assumes the following:

- the number of bar's entered is the number of bars displayed
- the inner window the indicator will be plotted in is square, or nearly so
- no other data than the last bars in the chart are displayed, if more are shown from an earlier time, i.e. scrolling back, the height calculation would be skewed.

These restrictions have generally made such a formula not usable.

Channels

The last line study to be looked at is the channel. MetaStock includes a family of channels that all have a Raff Regression Line as the center line. These are the Raff Regression Channel, the Standard Deviation Channel, and the Standard Error Channel. Since the only difference is how the top and bottom channel lines are calculated, this text will look at the regression channel.

Exercise 5

Write a formula to calculate a Raff Regression Channel. The channel should be calculated over fifty periods of price data, ending ten bars before the end of the chart, that is the last ten bars are not included in the calculation of the channel. However, the channel should be drawn to the end of the chart.

Solution

Step 1: An Analysis

To draw this, the regression line must be calculated. It is done similarly to the trendlines, except, instead of calculating from the starting point, the line is calculated from the end point. The LinearReg() function will return the ending point of a regression line of a set number of periods. The LinRegSlope() will give the slope of the same line.

The outer lines of a Raff Regression Channel are calculated by measuring the distance of the high and low from the center regression line. The largest distance over the length of the channel is how far the upper and lower lines are from the center line.

LastValue() will be used frequently to calculate variables that must be defined at the start of the channel but can not be known till the end. Cum(1) will be used to index the correct starting and ending points.

Step 2: The Logic

No prompts are required for the this formula so the definitions of the center line can be started right away. First get the end points and the slope. The variable ebar is created to contain the number of the bar for the end point. This will also be the reference point for all other bar number calculations:

```
ebar:= LastValue(Cum(1)-10);  
evalue := LastValue(ValueWhen(1, Cum(1)=ebar, LinearReg(C, 50)));  
slope := LastValue(ValueWhen(1, Cum(1)=ebar, LinRegSlope(C,50)));
```

Now the calculation of the center line is possible. The formula is the same as for previous line formulas, but notice, the number of bars from the fixed point is a negative number.:

```
center := evalue + (slope * (Cum(1) - ebar));
```

There is a new consideration this time. All the values for the line were calculated with the LastValue() function. They will be defined for every bar in the chart. BarsSince() is not being used so the line will be drawn through the entire chart. The requirements are for the formula to only draw the channel over the 50 bars it is calculated on plus the last 10 bars. To do that, the formula must be made undefined prior to the start of the channel.

Putting BarsSince() back in the formula is the simplest way to accomplish this. The function is added as the conditional statement of an If(). The slope calculation is the True result. A False result is required by the formula language syntax. Since the If() will be undefined if it is False, the value chosen does not matter. Zero is used below.

```
center := If(BarsSince(Cum(1) >= ebar-49) > -1,
  evalue + (slope * (Cum(1) - ebar)), 0);
```

We only subtract 49 from the location of ebar because ebar is the end point. we only draw the line over ebar and the 49 bars before it, not 50.

As soon as barsSince() finds where Cum(1) equals ebar-49, it will have a value. Depending on the mathematical comparison of Cum(1) to ebar-49, the value will be either zero or a positive number. It is also at that point that the variable center will become defined and can be calculated. Since BarsSince() will either be zero or higher, we just look for it to be greater than -1.

To calculate the distance for the upper and lower lines, the formula must find the furthest price from the line. This is done by checking the distance of the High from the center line on each point in the channel. Similarly, the distance between the Low and the center line is also measured. The largest value of each distance is found and assigned to the variables topdis and botdis. The largest of these two values is the distance of the channel lines from the center line.

```
topdis:= LastValue(ValueWhen(1, Cum(1)=ebar, HHV(H-center, 50)));
botdis:= LastValue(ValueWhen(1, Cum(1)=ebar, HHV(center-L, 50)));
dis:=Max(topdis,botdis);
```

Step 3: The Final Formula

To finish the channel, just call the center variable and add or subtract dis from the center line:

```
ebar:= LastValue(Cum(1)-10);
evalue := LastValue(ValueWhen(1, Cum(1)=ebar, LinearReg(C, 50)));
slope := LastValue(ValueWhen(1, Cum(1)=ebar, LinRegSlope(C,50)));
center := If(BarsSince(Cum(1) >= ebar-49) >-1,
  evalue + (slope * (Cum(1) - ebar)), 0);
topdis:= LastValue(ValueWhen(1, Cum(1)=ebar, HHV(H-center, 50)));
botdis:= LastValue(ValueWhen(1, Cum(1)=ebar, HHV(center-L, 50)));
dis:=Max(topdis,botdis);
center + dis;
center;
center - dis
```

Plotted on a chart, it would look like this:



The Standard Error and Standard Deviation formulas get their distance by calculating the standard deviation or standard error, as appropriate, over the periods the center line is drawn over. That value is then multiplied by the desired number of deviations or errors.

Chapter 6: The Logical Switch

The third chapter of the formula primer discussed binary waves when explaining If() functions. If() functions are the only decision making logic available to MetaStock formulas, yet used properly, they can do a lot more than is immediately obvious. Cue the logical switch. In its basic form, it is just a binary wave.

To review, a binary wave is a line that is either True or False. When it is True it usually has the value of 1. When False, it normally has the value of 0. Binary is a numerical counting system used by computers that only uses 1's and 0's. Hence the name.

But notice, the above explanation said the values of 1 and 0 were normally and usually used. They are not required. Consider the formula below:

```
C > Mov( C, 40, S)
```

This is a conditional statement. In a formula it would be evaluated and determined to be either True or False. Without any other code, MetaStock will interpret this to be identical to the formula below:

```
If( C > Mov( C, 40, S), 1, 0)
```

If the formula is written out, other values can be used besides 1 and 0. However, the true strength of the logical switch has not been revealed yet.

Two state switches

A switch is something that moves between two states. This could be a light turning on and off. However, it could also be a railroad track junction changing from the main line to a side track. It could be a radio changing from AM to FM reception.

In a formula, each state must be defined. It could be a single event like the close crossing above a moving average or volume exceeding some minimum level. This example will use the condition of the MACD being above or below its signal line. Each state will represent a time when a system would be either long or short.

```
Long := MACD() > Mov( MACD(), 9, E);  
Short := MACD() < Mov( MACD(), 9, E);
```

Since the system is either long or short, it would be nice to use 1 for long and -1 for short. This can not be done with just one If(). However unlikely it may be, there is a chance the MACD may equal its signal line. If it does, this switch should not change position until an actual cross occurs. To allow for that, two If() statements are nested together:

```
If( long, 1, If( short, -1, 0) )
```

However, this will still not work right because a third possible value is listed. The 0 needs to be removed and the formula must retain the value from the last true conditional statement. PREV does this very well. The final formula looks like this:

```
Long:= MACD() > Mov( MACD(), 9, E);  
Short := MACD() < Mov( MACD(), 9, E);  
If( long, 1, If( short, -1, PREV) )
```

The formula could use BarsSince() instead of PREV. Just change the last line to this:

```
If( BarsSince(long) < BarsSince(short), 1, -1)
```

This only uses one If() but there is a trade off. BarsSince() is undefined until the event it looks for occurs. This means the switch is undefined until both the long and short conditions have each happened at least once.

Why uses switches

It is possible the above just looks like a cute trick with the formula language. Do not be fooled. The above logic is a critical concept. This is the key that unlocks formulas which otherwise would be impossible. Here is a simple example to start with.

Exercise 1

MetaStock ships with a system test named “Equis - Moving Average Crossovers w/Opt”. The buy and sell signals are listed below.

Buy:

```
Mov(C, OPT1, E) > Mov(C, OPT2, E)
```

Sell:

```
Mov(C, OPT1, E) < Mov(C, OPT2, E)
```

To use these in an expert adviser, the opt variables would have to be replaced with constant values. Setting OPT1 to 15 and OPT2 to 40, the formulas would then become:

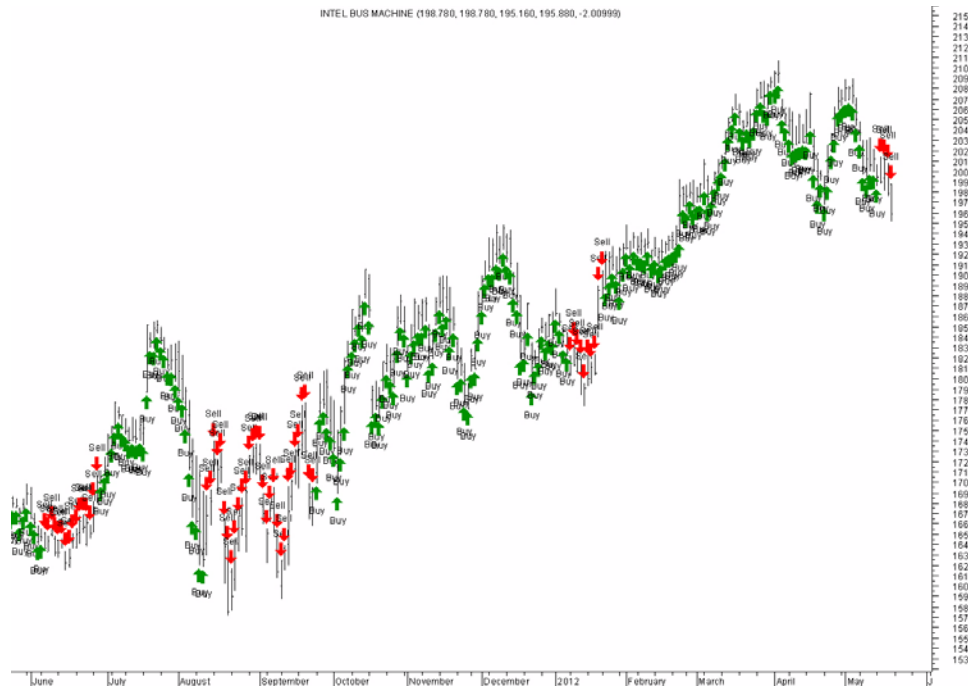
Buy:

$\text{Mov}(C, 15, E) > \text{Mov}(C, 40, E)$

Sell:

$\text{Mov}(C, 15, E) < \text{Mov}(C, 40, E)$

If these were used as is and put in an expert advisor to plot buy and sell arrows, the results would be similar to the chart below:



Use the logical switch to rewrite the buy and sell signals so only one arrow is placed per trade

Solution

Step 1: An Analysis

The current buy and sell signals can replace the formulas for the events. This will cause the switch to change value when the conditions change.

Step 2: The Logic

First the conditions are defined for the long and short positions:

```
long := Mov(C, 15, E) > Mov(C, 40, E) ;  
short := Mov(C, 15, E) < Mov(C, 40, E) ;
```

The switch can be copied as is from the example code:

```
If( long, 1, If( short, -1, PREV) );
```

A simple Ref() function can find when the current value of the switch is different than the value on the previous bar. If the switch is assigned to a variable, this part is easier to write. The buy signal check should look similar to the one below:

```
switch:=If( long, 1, If( short, -1, PREV) );  
switch = 1 AND Ref(switch <> 1, -1)
```

The sell signal check would be the same except the 1 is replaced with -1:

```
switch = -1 AND Ref(switch <> -1, -1)
```

Step 3: The Final Formula

Combining all the above code, the final signal formulas would look like the ones below:

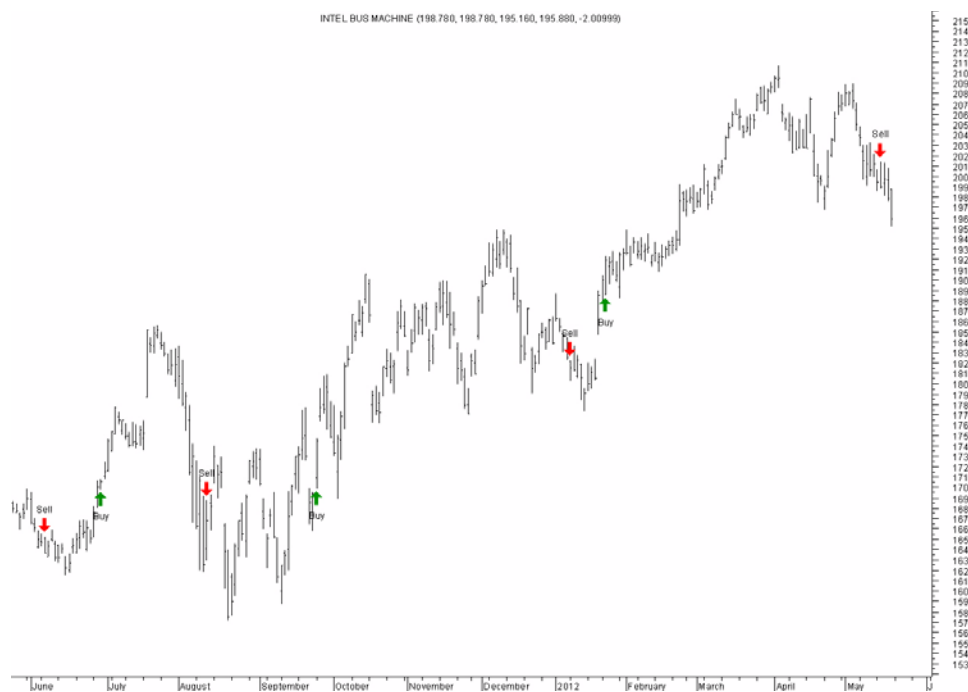
Buy:

```
long:= Mov(C,15,E) > Mov(C,40,E);  
short := Mov(C,15,E) < Mov(C,40,E);  
switch:=If( long, 1, If( short, -1, PREV) );  
switch = 1 AND Ref(switch <> 1, -1)
```

Sell:

```
long:= Mov(C,15,E) > Mov(C,40,E);  
short := Mov(C,15,E) < Mov(C,40,E);  
switch:=If( long, 1, If( short, -1, PREV) );  
switch = -1 AND Ref(switch <> -1, -1)
```

Now the expert advisor's chart looks this way:



Three state switches

A switch does not have to just have two positions and neither does this logical construction. By adding more If() functions, the switch can hold more values. It is a matter of nesting the If()s before the final PREV. Consider the following logic:

```
e1 := Mov(C,15,E) > Mov(C,40,E); {enter long}
es := Mov(C,15,E) < Mov(C,40,E); {enter short}
xl := Mov(C,5,E) < Mov(C,20,E); {exit long}
xs := Mov(C,5,E) > Mov(C,20,E); {exit short}
switch := If( e1, 1, If( es, -1,
  If( (PREV=1 AND xl) OR (PREV=-1 AND xs), 0, PREV) ) );
switch
```

Four conditional statements are provided. The first part of the switch still checks for the enter long and enter short conditions. However, instead of the final PREV, another If() has been nested. This one checks for two compound conditions:

- the switch had the value of 1 on the previous bar and now the exit long condition is true
- the switch had the value of -1 on the previous bar and now the exit short condition is true

If either is true, the switch takes the value of zero showing the system is out of the market.

But wait there's more

In Chapter 4, “Simulation Data”, it was noted there was a better way to get the entry price. That way is the logical switch. Instead of having the switch take the value of 1 when the buy signal is true, it can take the value of the closing price. This could be used to add additional positions to a trade.

Exercise 2

A system enters a long trade when the price rises above a 40-period simple moving average. The trade is closed when the price falls below the same average. Every 2% increase in price since the last order will add another position to the trade. All positions will be entered at the close of the signal bar. No short positions are in this system. Write the entry and exit signals for this system.

Solution

Step 1: An Analysis

This switch will have the usual entry and exit conditions. If the entry signal is True, the switch will take the value of the closing price. This is just a small modification to the previous code. However, the add-on entry signals, require a third If() to check for the subsequent entry signals.

Step 2: The Logic

Define the entry and exit conditions first:

```
e1 := Cross(C, Mov(C,40,S)); {enter long}
x1 := Cross(Mov(C,40,S), C); {exit long}
```

The switch will start with two nested If(s). The enter long signal will set the switch to the close of the current bar. The exit long signal will set the switch to zero to show the system is out of the market.:

```
switch := If( e1, C, If( x1, 0,
```

As the three state switch checked PREV to verify the exit signal was for the correct market position, here PREV holds the price of the last entry signal. By comparing the current price to PREV, new signals can be generated. First it checks to verify the switch is greater than zero, indicating a trade is in progress. Then it compares the close to the previous value of the switch times 1.02 (2% higher than the last entry price). If both conditions are True, the switch is reset to the value of the current close

```
If( PREV>0 AND C>= PREV*1.02, C, PREV) );
```


Step 3: The Final Formula

Using this logic the final switch and signal formulas would be:

enter long

```
el := Cross(C, Mov(C,40,S));
xl := Cross(Mov(C,40,S), C);
switch := If( el, C, If( xl, 0,
  If( PREV>0 AND C>= PREV*1.02, C, PREV)));
switch > Ref(switch, -1)
```

exit long:

```
el := Cross(C, Mov(C,40,S));
xl := Cross(Mov(C,40,S), C);
switch := If( el, C, If( xl, 0,
  If( PREV>0 AND C>= PREV*1.02, C, PREV)));
switch = 0 AND Ref(switch <> 0, -1)
```

Notice the final line of the buy signal is slightly different this time. Now it looks for anytime the switch has a value greater than it had on the previous bar. The only set value for the switch is zero. All other values will be the close of the signal bar so they can not be referenced directly. However, each new position will always have a higher close than the one before.

Counting positions

Another use of the logical switch is to limit the number of new positions. The previous logic allowed positions to be added until the sell signal was found. Suppose money management rules restricted this to just four positions.

Exercise 3

All prior conditions are the same except there can only be four positions per trade.

Solution

Step 1: An Analysis

Sometimes one switch is not enough. The logic gets too complex. This formula will use two switches.

Step 2: The Logic

The first switch is the same as from the previous exercise.

```

el := Cross(C, Mov(C,40,S));
xl := Cross(Mov(C,40,S), C);
switch := If( el, C, If( xl, 0,
    If( PREV>0 AND C>= PREV*1.02, C, PREV)));

```

The entry and exit signals are now used to make the variables bsig and ssid.

```

bsig := switch > Ref(switch <> 1, -1);
ssid := switch = 0 AND Ref(switch <> 0, -1);

```

A second switch is named limit. It looks for the buy signal and verifies the current value of limit is less than 4. If both conditions are True, limit is set to its previous value plus 1. The sell signal resets limit to the value of zero

```

limit := If( bsig AND PREV < 4, PREV +1,
If( ssid, 0, PREV));

```

Step 3: The Final Formula

The final lines use the same logic as in the previous exercise. This time, they look at the variable limit instead of switch:

Buy:

```

el := Cross(C, Mov(C,40,S));
xl := Cross(Mov(C,40,S), C);
switch := If( el, C, If( xl, 0,
    If( PREV>0 AND C> PREV*1.02, C, PREV)));
bsig := switch > Ref(switch, -1);
ssid := switch = 0 AND Ref(switch <> 0, -1);
limit := If( bsig AND PREV < 4, PREV +1,
If( ssid, 0, PREV));
limit > Ref(limit,-1)

```

Sell:

```

el := Cross(C, Mov(C,40,S));
xl := Cross(Mov(C,40,S), C);
switch := If( el, C, If( xl, 0,
    If( PREV>0 AND C> PREV*1.02, C, PREV)));
bsig := switch > Ref(switch, -1);
ssid := switch = 0 AND Ref(switch <> 0, -1);
limit := If( bsig AND PREV < 4, PREV +1,
If( ssid, 0, PREV));
limit = 0 AND Ref(limit <> 0, -1)

```

There are limits

Theoretically, there is no limit to how complex a switch could be made. By nesting If() functions, multiple paths and branches can be made. However, only one value can be carried from the previous bar. When designing the switch, what will be carried must be kept in mind.

A second limitation is much more nebulous. PREV functions add memory requirements and processing time. A single use will not be noticeable, but multiple uses increase this. While two and three are safe, four or more will start showing noticeable delays. Eventually, the formula will cease to calculate.

If() functions can also cause issues. The more levels they are nested in, the more complex the calculation appears to MetaStock. This can cause a slowdown in performance. Eventually, it becomes so complex, MetaStock gives a “binary too complex” error (or similar message) and refuses to plot anything with the formula.

There is no set number of If() functions that are too many. It is a matter of how they are arranged. Nested one way and only five or six If()s will cause the error. Arranged another way and a formula might be able to use seven or eight. The number of PREVs mixed in with the switch will also affect it and limit the complexity of the If()s. The only way to know if something is possible is to try it. If it does not work, try rearranging the switch structure. A different arrangement may work better.

Chapter 7: Pivot Points

Almost everyone who has studied technical analysis has heard the term pivot points. They are discussed in several books and numerous magazine articles. What is not obvious is all these texts are not necessarily discussing the same thing. This chapter will examine three different types of pivot points

Middle of the road

The name "pivot" suggests a point around which something will turn or rotate. The first pivot method calculates the mid point of the previous day's trading and uses it to plot support and resistance values into the next day. The theory is that the prices will not move too far from the previous day's trading. The midpoint is the expected pivot around which the next day's trading will develop.

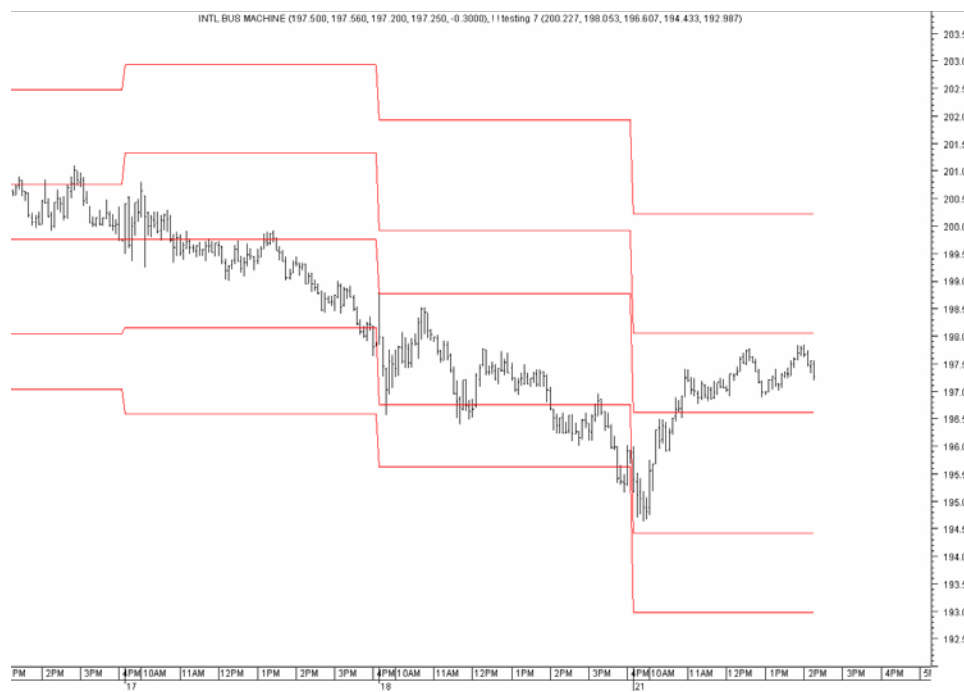
There are even variations on this type of pivot. However, the differences are on how the support and resistance lines are calculated. The pivot point is the same; the average of the previous day's high, low, and closing prices. This value is also known as the Typical Price. The trick is finding the daily value from intraday charts.

Exercise 1

You have a chart with five minute data for a stock. Calculate the pivot point of the previous day and plot it as a line through the next day. Additionally, plot the first and second support and resistance lines. For this exercise, those lines will be defined as:

S1 = two times the pivot point minus the high
R1 = two times the pivot point minus the low
S2 = pivot point plus S1 minus R1
R2 = pivot point plus R1 minus S1

The chart with those lines plotted on it will look similar to the one below



Solution

Step 1: An Analysis

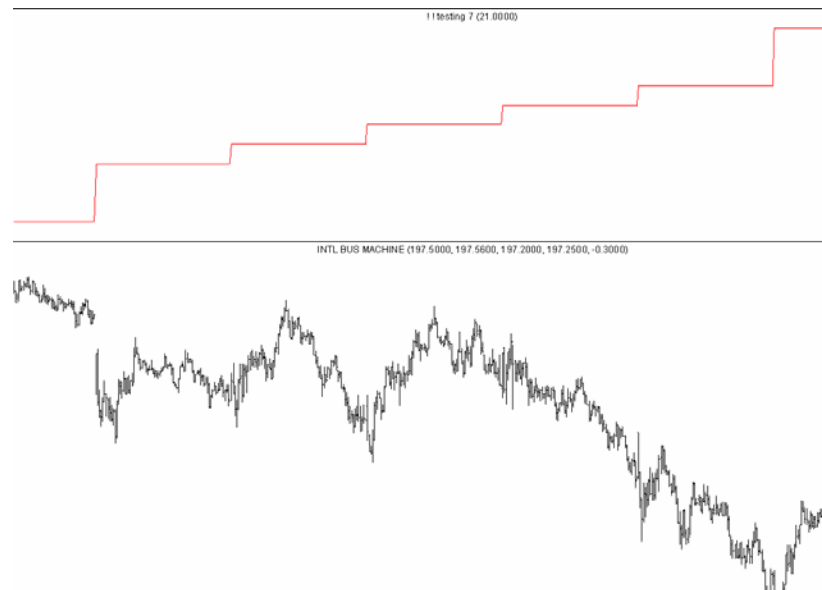
There is no function to get daily values when an intraday chart is open. Therefore, the existing functions will need to be employed to calculate the daily high, low and close.

HighestSince(), and LowestSince() can find the high and low over an indeterminate span of time. They only require an event to measure from. These functions will also include all the data to the current bar so more than this is required.

ValueWhen() can return the value of some data array or function from some defined event earlier in the data.

If the event is defined as the start of the trading day, then we almost have everything we need for the formula. We just need to decide how to identify a new day. The Hour() and Minute() functions are one way. However, different exchanges start trading at different times. An alternative is the

DayofWeek() and DayofMonth() functions. If these are plotted in an intraday chart, they show a flat line that only changes when a new day starts



Step 2: The Logic

We will define our event as the first bar of a new day. The new variable below identifies this bar by a change in value of the DayofMonth function:

```
new:= DayofMonth() <> Ref( DayofMonth(), -1);
```

We can then nest that inside the ValueWhen() function to find the close of the previous day. The variable "new" identifies the bar a new day starts on so ValueWhen() has to include Ref() to get the value of the previous bar's closing. To keep the name of the variable short, we will use yc for yesterday's close

```
yc := ValueWhen( 1, new, Ref( C, -1) );
```

By replacing the close with the HighestSince() and LowestSince() functions, it is possible to get the high and low of the previous day. We will use yh and yl for those variables:

```
yh:= ValueWhen( 1, new, Ref( HighestSince( 1, new, H), -1) );
yl:= ValueWhen( 1, new, Ref( LowestSince( 1, new, L), -1) );
```

Now that the high, low, and close of the previous day are defined the pivot point is easy. Add yc, yh, and yl together and divide the result by 3. we will assign this to the variable pp:

```
pp:= (yc+yh+yl) /3;
```

Step 3: The Final Formula

The rest of the formula is basic math. The formula below assigns the support and resistance lines to variables and then calls the variables in top to bottom order. This is done so a tool tip on the indicator will list the lines from highest to lowest.

```
new:=ROC(DayOfWeek(),1,$)<>0;
yh:=ValueWhen(1,new, Ref(HighestSince(1,new,H),-1));
yl:=ValueWhen(1,new, Ref(LowestSince(1,new,L),-1));
yc:=ValueWhen(1,new, Ref(C,-1));
pp:=(yc+yh+yl)/3;
r1:=(pp*2)-yl;
s1:=(pp*2)-yh;
r2:= pp+r1-s1;
s2:= pp-r1+s1;
r2;
r1;
pp;
s1;
s2
```

Highs and Lows

Another common definition of a pivot point is the high and low points of a price move. This could also be called the peaks and troughs. These are the most difficult to define with a formula. They are easily picked out with the naked eye because we can see the prices changed direction. However a formula does not have the same capabilities of the human brain. A formula requires a mathematical test it can apply to the data.

The Peak() and Trough() functions will find these types of pivots, but they require a predetermined reversal amount. A value that is perfect for one chart may not find anything on another chart, or perhaps worse, be too sensitive and bounce around when the prices are actually flat and consolidating. A more creative solution is called for.

Exercise 2

One method is to look for a new high or a new low over some medium length of time, say 40 bars. This will identify an upward move of the prices above a recent dip or flat area. However it does not identify the reversal. Therefore, a secondary condition is needed. Look for the prices to drop on the bar after the new high. This is for an expert advisor so have the formula be true on the bar that forms the peak

Solution

Step 1: An Analysis

This formula is not much more complicated than those in the formula primer. It is just two conditions with a time element requiring them to occur in a specific sequence.

Step 2: The Logic

The new high will occur on the bar the we want the signal on. The drop in price will happen one bar after the signal, so it needs to be checked for inside a Ref() function using a +1 reference

Step 3: The Final Formula

$H > \text{Ref}(\text{HHV}(H, 40), -1) \text{ AND } H > \text{Ref}(H, +1)$

When this is used in an expert, it can generate several signals in a uptrend, but as the prices decline, it can be referenced to find the peak of the last up move. This is illustrated in the chart below:



Further refinements could be made to reduce the false signals. There are many possibilities and each one will have its own strengths and weaknesses.

Parts of a Whole

Bill Williams has published several articles and books on using fractals with technical analysis. It is beyond the scope of this text to cover all his work. However, he has defined a fractal (also termed a pivot) high and low as a series of bars that form a specific pattern. What distinguishes his pattern is the number of variations. If any of the possible variations is met, the pattern is seen as being true. Below is a visual representation of the various fractal highs

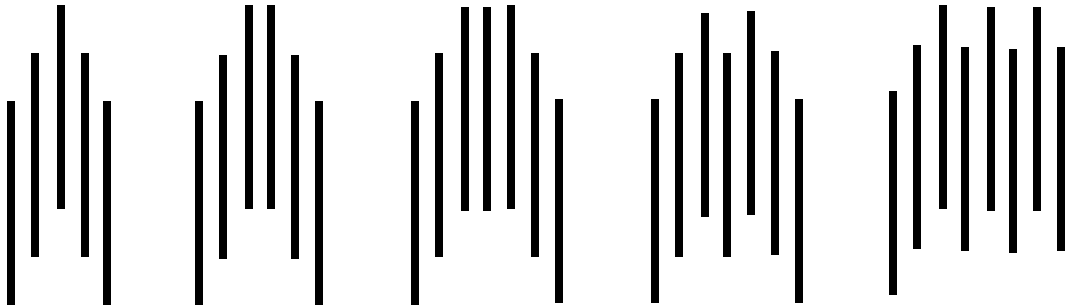


Figure 7.1: Fractal High Patterns

Exercise 3

Write a formula to find a fractal high. It needs to recognize all 5 variations. The formula is for a trade entry signal so it should not use any references to future data.

Solution

Step 1: An Analysis

Treat this as 5 different pattern formulas. Assign each pattern to a variable whose formula is TRUE when that pattern is found. All calculations can use negative references values only to avoid using future data

Step 2: The Logic

The first pattern is a basic 5 bar pivot high. It does not matter how much higher the middle bar is than the two to either side of it. It does not matter if the first and last bars have lower highs than the two closer to the middle. All that matters is that the middle bar has the highest high of the five bars:

```
bar1:= Ref (H, -2) > HHV (H, 2) AND  
Ref (H, -2) > Ref (HHV (H, 2) , -3) ;
```

The second pattern is the same except it uses 6 bars and the middle two have identical highs:

```
bar2:= Ref (H, -2) > HHV (H, 2) AND
Ref (H, -2) = Ref (H, -3) AND
Ref (H, -2) > Ref (HHV (H, 2) , -4) ;
```

The third pattern continues the theme and says the middle three bars all have the same high:

```
bar3:= Ref (H, -2) > HHV (H, 2) AND
Ref (H, -2) = Ref (H, -3) AND
Ref (H, -2) = Ref (H, -4) AND
Ref (H, -2) > Ref (HHV (H, 2) , -5) ;
```

The last two require a bit more effort. pattern four is seven bars, but bars 3 and 5 have the same high and the middle bar has a lower high. The “nc” in the variable name stands for non-consecutive:

```
bar2nc:= Ref (H, -2) > HHV (H, 2) AND
Ref (H, -2) > Ref (H, -3) AND
Ref (H, -2) = Ref (H, -4) AND
Ref (H, -2) > Ref (HHV (H, 2) , -5) ;
```

The final pattern is nine bars long. The 3rd, 5th, and 7th should all have the same high with all other bars having lower highs:

```
bar3nc:=Ref (H, -2) > HHV (H, 2) AND
Ref (H, -2) = Ref (H, -4) AND
Ref (H, -2) = Ref (H, -6) AND
Ref (H, -2) > Ref (H, -3) AND
Ref (H, -2) > Ref (H, -5) AND
Ref (H, -2) > Ref (HHV (H, 2) , -7) ;
```

Step 3: The Final Formula

The final formula combines all the variables and then follows that with condition using OR so it is TRUE if any of the variables are true.

```
bar1:= Ref (H, -2) > HHV (H, 2) AND
Ref (H, -2) > Ref (HHV (H, 2) , -3) ;
```

```
bar2:= Ref (H, -2) > HHV (H, 2) AND
Ref (H, -2) = Ref (H, -3) AND
Ref (H, -2) > Ref (HHV (H, 2) , -4) ;
```

```
bar3:= Ref (H, -2) > HHV (H, 2) AND
Ref (H, -2) = Ref (H, -3) AND
Ref (H, -2) = Ref (H, -4) AND
Ref (H, -2) > Ref (HHV (H, 2) , -5) ;
```

```

bar2nc:= Ref (H, -2) > HHV (H, 2) AND
Ref (H, -2) > Ref (H, -3) AND
Ref (H, -2) = Ref (H, -4) AND
Ref (H, -2) > Ref (HHV (H, 2) , -5) ;

bar3nc:=Ref (H, -2) > HHV (H, 2) AND
Ref (H, -2) = Ref (H, -4) AND
Ref (H, -2) = Ref (H, -6) AND
Ref (H, -2) > Ref (H, -3) AND
Ref (H, -2) > Ref (H, -5) AND
Ref (H, -2) > Ref (HHV (H, 2) , -7) ;

bar1 OR bar2 OR bar3 OR bar2nc OR bar3nc

```

Building blocks

Fractals are used in graphics, math and science. They are like Legos. The basic idea is a small pattern that repeats and builds upon itself. Each time it is duplicated, the whole becomes more elaborate and complex. The end result can be an exotic structure or a jagged landscape. However, no matter how complex the final pattern is, it is still composed of nothing but repetitions of the same basic pattern. The pattern may be twisted, rotated or turned completely around, but it is still there if we look for it.

Golson's Pivots took this idea and a basic version of Bill Williams fractals. The minor pivot was defined as any bar with a high and a lower high on both sides of it. This forms an upside down V pattern. It does not matter what the trend is. This is one of the basic blocks. The other is the opposite pattern using the Low prices. The pivot is coded to be true on the high/low of the pivot point. The formulas are listed below:

Golson's Minor Pivot High

```
H > Max ( Ref (H, -1) , Ref ( H, +1) )
```

Golson's Minor Pivot Low

```
L < Min ( Ref (L, -1) , Ref ( L, +1) )
```

Exercise 4

The fractal part is how the Intermediate and Major pivots are built. An intermediate pivot high is Any pivot high that has a lower pivot high on both sides of it. Thus if you were looking at just the pivot highs, the three pivots would again form any inverted V pattern. Like the minor pivot, the intermediate pivot should be true on the high of the central pivot pattern.

Solution

Step 1: An Analysis

Like the minor pivot pattern, the intermediate pivot will require looking into the future to correctly mark the pivot high. Unlike the minor pivot, there is not a set number of bars to look for. The next pivot high could be as few as two bars into the future or it could be a hundred. Usually, it will be closer to the first, but the number is unknown when writing the formula. The only solution is to query the data for the number of bars to look ahead.

The LastValue() function will calculate the value inclosed inside the parenthesis as if it was on the last bar in the chart. That can be used to find the longest time between minor pivots. Then, that value can be used as the forward reference.

Step 2: The Logic

Start by defining the basic block, the minor up pivot. We will set this to the variable mu.

```
mu:= H > Max( Ref(H, -1), Ref(L, +1) );
```

The left side of the pivot will require the current bar to be a pivot and the previous pivot to have a lower high:

```
p2:= mu AND H>ValueWhen(2,mu,H);
```

The right side of the pivot, when examined from the last pivot in the pattern will consist of a pivot with a lower high than the previous pivot AND the previous pivot must look like the left side of the pattern (i.e.; it must have a lower high to the left of it):

```
p3:=mu AND H<ValueWhen(2,mu,H) AND ValueWhen(2,mu,p2);
```

BarsSince() can find the time since an event happened. We can nest that inside the Highest() function and nest that inside LastValue() for the time:

```
time:= LastValue(Highest(ValueWhen(1,p3,BarsSince(p2))));
```

Notice the ValueWhen() is specifically measuring how much distance is between a left side and a right side of the pivot pattern. Highest returns the longest time between the two points.

Since pivots can occur frequently, it is possible more than one pivot may occur during this time. One more step will be taken. We will use Cum(1) to number the bars (as was done on page 13). By subtracting the value of BarsSince() from this number, the bar number of a past event can be calculated. We will look into the future the number bars assigned to the time variable and apply this calculation to get the bar number of the intermediate pivot point. This value will be compared to the current bar number to determine when to signal the intermediate pivot.

Step 3: The Final Formula

```
mu:=H>Max(Ref(H,-1),Ref(H,1));
p2:=mu AND H>ValueWhen(2,mu,H);
p3:=mu AND H<ValueWhen(2,mu,H)
AND ValueWhen(2,mu,p2);
time:=LastValue(Highest(ValueWhen(1,p3,BarsSince(p2)))));
x:=ValueWhen(1,p3,Cum(1)-BarsSince(p2));
Cum(1)=Ref(x,time)
```

When used in an expert, you will get a chart like the one below. The circles are minor pivots, the diamonds are intermediate pivots. The exclamation marks are major pivots.



The major pivots use the same logic as the intermediate pivots. LastValue is combined with Highest() and BarsSince() to find the most time between intermediate pivots. This time is combined with ValueWhen() to get the future intermediate pivot high. The final formula is listed below

```
mu:=H>Max(Ref(H,-1),Ref(H,1));
p2:=mu AND H>ValueWhen(2,mu,H);
p3:=mu AND H<ValueWhen(2,mu,H)
AND ValueWhen(2,mu,p2);
time:=LastValue(Highest(ValueWhen(1,p3,BarsSince(p2)))));
x:=ValueWhen(1,p3,Cum(1)-BarsSince(p2));
mju:=Cum(1)=Ref(x,time);
pj2:=mju AND H>ValueWhen(2,mju,H);
pj3:=mju AND H<ValueWhen(2,mju,H)
AND ValueWhen(2,mju,pj2);
timej:=LastValue(Highest(ValueWhen(1,pj3,BarsSince(pj2)))));
```

```
xj:=ValueWhen(1,pj3,Cum(1)-BarsSince(pj2));  
Cum(1)=Ref(xj,timej)
```

Chapter 8: Logical Twists

As was mentioned in the formula primer, the Cathedrals of medieval Europe were designed with three basic tools, the straight edge, a compass, and a square. This chapter will introduce some tools that while not as universally usable, are the only way to resolve some issues. These are like some very specialized Lego bricks. These tools are, dynamic constants, normalized values, sequential constructions and divergences.

Dynamic constants

Many MetaStock functions have one or more parameters that must be a constant, like all references to time periods. Unfortunately, sometimes a value needs to be adjusted. For example, a formula may want to adjust the time periods of a moving average by adding half the value of the ADX to a base number of periods. Traditionally, this is not possible in MetaStock. However, there is a way.

This method does not always work. It is what developers might call a “kludge”. It takes advantage of the side effects of two functions to create a value the formula language will treat as a constant but which is actually recalculated on every bar.

The first function to be used is the last value function. As defined, it goes to the end all the loaded data and calculates whatever value is included in its parenthesis. The Raff Regression Channel formula used LastValue() in several places to define values needed for its calculations (See “Channels” on page 34). Consider this formula:

```
Bars:= LastValue( Cum( 1 ) );  
Periods := LastValue( If( bars >= 200, 200,  
    If( bars >= 40, 40, Max(bars-1,1)) ) );  
Mov( C, periods, S)
```

It uses LastValue to see how many bars of data are available. If the chart has two hundred or more bars loaded, it will calculate a two hundred period moving average. If there are less than two hundred but at least forty, it will calculate a forty period moving average. If there are less than forty bars of data, it takes the number of bars minus one, (with a minimum of one bar) and calculates an average of that many periods. With LastValue, the formula knows on the first bar of the chart how much data it has to work with.

The second function to be used is PREV. PREV was used in the logical switch to carry a value forward to the next bar. PREV does this by requiring the formula language to go back to the previous bar and recalculate the formula for that bar. This caused the formula to constantly loop back on itself. Consider the following showing how PREV affects a formula’s calculation:

- A formula uses a five period moving average with other data values like the High, Low, Close, and Volume. It also contains the PREV function.
- Prior to the fifth bar, the formula is undefined as the moving average can not be calculated.
- On the fifth bar, PREV has the value of zero as there is no previous value to refer to.
- While calculating the value for the sixth bar, the formula loops back and calculates the value it had on the fifth bar.
- When it calculates the seventh bar, it again loops back to calculate the value it had on the sixth bar.

This constant recalculation of the prior bar's value is why the PREV function slows down formulas. However, it has another effect as well. In the example above, when the formula loops back on the sixth bar to calculate the value of the fifth bar, LastValue will then see the sixth bar as the last bar loaded in the chart. On the seventh bar, as PREV causes the loop back LastValue() will then see the seventh bar as the last bar loaded.

To combine the effects, the PREV must be inside the parenthesis of the LastValue function. This means PREV needs to be used in some manner that will not alter the result of the value you want to dynamically assign. This could be done by first adding and then subtracting the value of PREV:

```
LastValue( ADX(14)/2 +PREV -PREV )
```

This is the original method for the kludge but it has since been refined to a single use of PREV:

```
LastValue( ADX(14)/2 +(PREV * 0) )
```

Since any number times zero is zero, adding PREV times zero to the value is the same as adding zero. However, the combination of PREV and LastValue() will still take effect:

```
Mov( C, LastValue( ADX(14)/2 +(PREV * 0) ), S)
```

This is the only way to do this sort of action with just the MetaStock formula language. An external DLL could do this, but it would need to be written specifically for each indicator a dynamic constant was to be used in.

Normalization

Sometimes there is a need to compare two things that do not share the same scale. Consider a chart with an On Balance Volume indicator plotted in the same inner window as an RSI. They both use their own scale and so appear to cross when mathematically they do not. It is theoretically possible these two indicators could give a valid signal when the cross happens. To find such signals, the scales of the indicators will have to be "normalized" to a shared base. The process to do this is:

1. Set the number of bars displayed in the inner window
2. Get the high and low of the indicators' scales for that number of bars
3. Subtract the low from the high of both scales to get the ranges

4. Divide the ranges of both scales by 100 to get a scaling factor for each indicator.
5. Subtract the scale low from each indicator and then multiply the result by its corresponding scaling factor. This normalize both to a zero to 100 scale.

Now, the formula can see if the two indicators are crossing each other. Those steps may be confusing to just read so here is a formula that actually does them

```
ti:= 250;
p1:= RSI(14);
p2:= OBV();

p1h:=HHV(p1,ti);
p1l:=LLV(p1,ti);
p1f:=100/(p1h-p1l);
p1m:= (p1-p1l)*p1f;

p2h:=HHV(p2,ti);
p2l:=LLV(p2,ti);
p2f:=100/(p2h-p2l);
p2m:= (p2-p2l)*p2f;

Cross(p1m, p2m) or Cross(p2m, p1m)
```

The first three lines set the number of bars displayed to 250 and specifies the two plots to be normalized are the RSI and the On Balanced Volume.

The next four lines gets the highest value and lowest value for the first indicator over the displayed range of data (variables p1h and p1l). The scaling factor is calculated in variable p1f and then the indicator is normalized in variable p1m. these steps are repeated for the second indicator in the four lines after that (variables p2h, p2l, p2f, and p2m).

The final line looks for the normalized data to cross either up or down.

Please understand, this cross may only be visible if the chart is displaying exactly 250 bars. Additionally, the bar of the cross must be the right most bar displayed. Change how many bars are displayed or scroll the chart to put the signal bar anywhere else but the right most bar and the formula will be using different data for the normalization than you are seeing. This will most likely cause a different scaling to take place visually and could alter when the chart shows a cross happening. If you force the chart to show all available data, and have the formula use the same, the signals will always match.

Sequential Constructions

Many signal patterns are actually a series of events that must occur in an exact order. For example a buy signal may have the following conditions:

1. The Close makes a forty bar high and is above its two hundred bar simple moving average.
2. The price then drops two percent or more from that high.
3. The price increases exceeding the previous new high.

These three events must occur in order or the signal is not valid.

Exercise 1

Write a formula that finds that signal. If the price falls below the 200 period moving average, the signal is void and is considered a false pattern.

Solution

Step 1: An Analysis

In these types of formulas, it is important to know if any condition or event will invalidate the pattern. In this case, a cross below the moving average is the reset condition.

Step 2: The Logic

Since time is not a factor, a modified logical switch will find this signal. First, it helps to define the conditions up front. By assigning them to variables, the logic of the switch is kept cleaner and easier to read:

```
reset := Cross( Mov( C, 200, S), C );
con1:= C > Ref( HHV( C, 40), -1) AND C> Mov( C, 200, S);
con2:= C <= ValueWhen(1, con1, C*0.98);
con3:= C > Ref(ValueWhen(1, con1, C), -1);
```

For the If, the reset should take precedence over any other signals, so it goes first. Afterwards, since the first and third condition are similar and could both be true at once, they will be checked in reverse order. Otherwise, the first condition could mask the third one.

```
bsig := If( reset OR PREV =3, 0,
If( con3 AND PREV=2, 3,
If( con2 AND PREV=1, 2,
If( con1, 1, PREV ))));
```

Now the formula just needs to look for the bsig variable taking the value of 3

Step 3: The Final Formula

```
reset := Cross( Mov( C, 200, S), C );
con1:= C > Ref( HHV( C, 40), -1) AND C> Mov( C, 200, S);
con2:= C <= ValueWhen(1, con1, C*0.98);
con3:= C > Ref(ValueWhen(1, con1, C), -1;
bsig := If( reset OR PREV =3, 0,
```

```

If( con3 AND PREV=2, 3,
If( con2 AND PREV=1, 2,
If( con1, 1, PREV ))));
bsig = 3 AND Ref(bsig<>3, -1)

```

The first line of bsig included a check of the signal already equalling three. This was done so the signal would reset after giving a positive. That way, if the system was to grab a quick profit and then wait for the signal to repeat, it would not have to wait for the price to drop below the moving average. In a protracted uptrend, this pattern could happen several times.

If time was a concern, BarsSince() can be used to check the time since the first condition. The logic could have been expressed completely with BarsSince() this way:

```

reset := Cross( Mov( C, 200, S), C );
con1:= C > Ref( HHV( C, 40), -1) AND C> Mov( C, 200, S);
con2:= C <= ValueWhen(1, con1, C*0.98);
con3:= C > ValueWhen(1, con1, C);
step1:= con2 AND BarsSince(con1) < BarsSince(reset);
bsig := con3 AND BarsSince(step1) < BarsSince(reset);
bsig = 1 AND Ref(bsig<>1, -1)

```

In this version, bsig takes the value of 1 (or TRUE) instead of 3. Also, an intermediate variable, step1, was added. This formula has the merit of not using PREV. Complex patterns that have many steps may require more uses of PREV than the formula language can handle. It will also calculate faster.

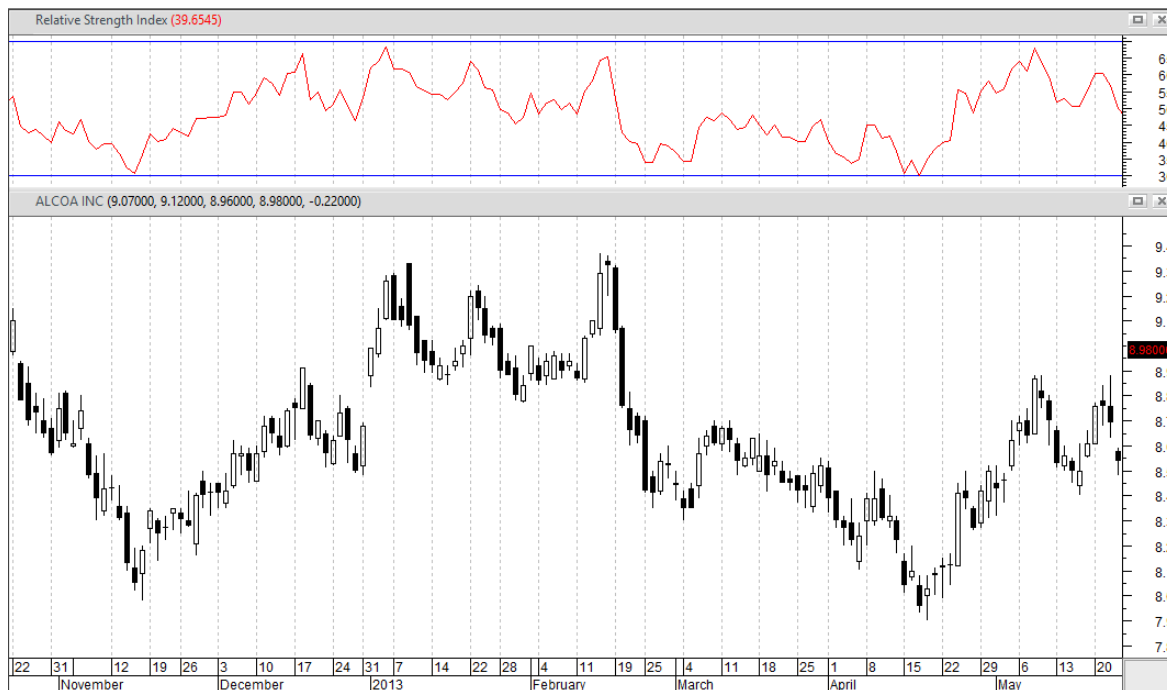
However, the formula will be undefined until the variables reset, con1, con2, and step1 have all been TRUE on at least one bar. This is the biggest detractor of the BarsSince() method.

Divergences

Divergence is a wonderful term to describe something the eye easily sees but which takes more effort to define formulaically. Further, the measurement of a divergence can vary. Consider a typical bullish divergence. The traditional definition is an indicator making higher lows while the prices make lower lows. This is not sufficient information to create a formula. The following questions must still be answered:

- How is a low defined?
- Are the lows of the prices and the indicators measured separately? That is, if the prices make a low on the march 5, does the formula look at the value of the indicator on March 5 or does it look for the low of the indicator?
- Do all the lows have to occur within a certain time of each other?

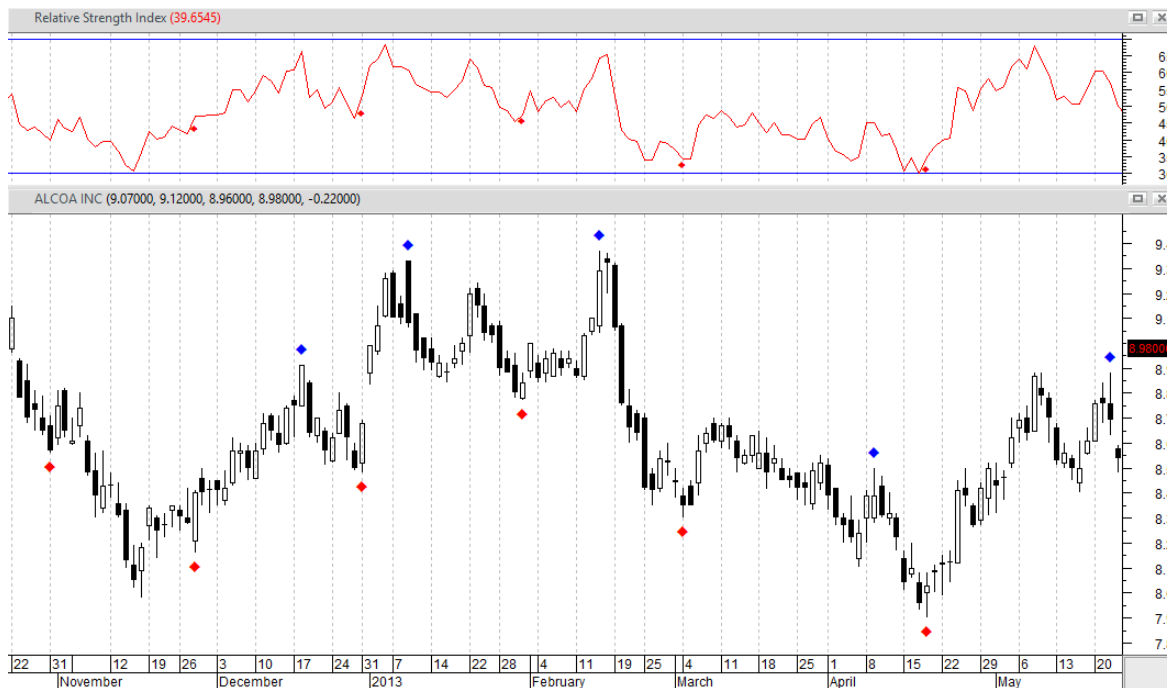
How these questions are answered directly affects which patterns are found. Consider the following chart of Alcoa:



If the lows were selected based on a 5% zig zag of the prices and a separate 5% zig zag of the indicator, they could be several weeks apart. This is where the time issue will be most important.



Alternately if the price lows were defined by a variation of Golson's pivots (page 54) and the indicator's value at the time of the price pivot was used, entirely different patterns will be found:



Regardless of how the questions are answered, the basic logic is still the same.

1. First the low is defined.
2. Then the value of the prices are taken at the two most recent lows.
3. If necessary, the lows of the indicator is defined
4. The indicator is measured at it's last two low points.
5. The values are checked to see if the indicator's most recent low is higher than the previous one and the most recent price low is less than its previous one.

Exercise 2

Write a formula to look for a bearish divergence between the prices and 14 period RSI. The first peak of the RSI must be greater than 70 though the second one does not have to be. For the peak definition, look for a bar that has four lower values to the left and one lower value to the right. Find the price and indicator peaks separately. It is desirable that the peaks be close to one another so there should be no more than a month of data between the first and last peak of this pattern.

Solution

Step 1: An Analysis

A bearish divergence is just the opposite of a bullish one, the indicator is making lower highs while the prices are making higher highs. The peak definition is a variation on the five bar pivot.

ValueWhen() with Cum(1) should be used to find which peak was the earliest. A typical month has twenty-two bars so the earliest peaks should not be farther away than that.

Step 2: The Logic

First the peaks are defined:

```
i1:= RSI(14);  
ip:= I1 < Ref(i1,-1) AND Ref(i1,-1) < Ref(LLV(i1,4),-2);  
pp:= H < Ref(H,-1) AND Ref(H,-1) < Ref(LLV(H,4),-2);
```

The indicator was assigned to a variable because it makes changing the formula later easier and it is faster to type I1 than the indicator formula.

Now the values of the four peaks and the bar number of the first two peaks will be defined. Be sure to use Ref(-1) to get the values since the peak variables are true on the bar after the peak.

```
ip1:= ValueWhen(1, ip, Ref(i1,-1));  
ip2:= ValueWhen(2, ip, Ref(i1,-1));  
itime:= Cum(1) - ValueWhen(2, ip, Ref(Cum(1),-1));  
pp1:= ValueWhen(1, pp, Ref(H,-1));  
pp2:= ValueWhen(2, pp, Ref(H,-1));  
ptime:= Cum(1) - ValueWhen(2, pp, Ref(Cum(1),-1));
```

Step 3: The Final Formula

All that remains is to compare the values. This give a finished formula of:

```
i1:= RSI(14);  
ip:= I1 < Ref(i1,-1) AND Ref(i1,-1) < Ref(LLV(i1,4),-2);  
pp:= H < Ref(H,-1) AND Ref(H,-1) < Ref(LLV(H,4),-2);  
ip1:= ValueWhen(1, ip, Ref(i1,-1));  
ip2:= ValueWhen(2, ip, Ref(i1,-1));  
itime:= Cum(1) - ValueWhen(2, ip, Ref(Cum(1),-1));  
pp1:= ValueWhen(1, pp, Ref(H,-1));  
pp2:= ValueWhen(2, pp, Ref(H,-1));  
ptime:= Cum(1) - ValueWhen(2, pp, Ref(Cum(1),-1));  
ip1 < ip2 AND pp1 > pp2 AND itime <= 22 AND ptime <= 22
```

Chapter 9: Custom Stops

The stops built into MetaStock are limited in their ability to be customized and are only available in the system tester. The expert advisor can not use them to display symbols on charts and explorations cannot search open positions for stop signals. For those actions, the stops have to be coded as custom formulas.

When coding a stop, the formula must know the point in time the stop is calculated from. This can be imputed for indicators but otherwise must be a condition written in the formula. The formula also needs a condition to restart the stop so a new trade can be tracked. Finally, if there are other exit conditions besides the stop, those must be include as well or the stop may give a signal after the trade has already been closed. The logical switch is the most easily adapted to track these multiple conditions.

Profit target

The profit target is probably one of the easiest to code. It is a set percentage or number of points from the entry price. When that point is reached, the trade is exited.

Exercise 1:

Add a 10 point profit target stop to a system that enters and exits on the close crossing above and below a 40 period moving average.

Solution

Step 1: An Analysis

This can be done with a three state switch. Use the sequential logic structure, page 60, as the example to start from.

Step 2: The Logic

Start by defining the entry and exit conditions. A variable is also added at the top for the size of the stop. This makes any future changes to the value easier.

```
profit:= 10;  
bsig:= Cross( C, Mov(C, 40, S));  
ssig:= Cross( Mov(C, 40, S), C);
```

Now the switch logic is coded. the entry price will be carried by the PREV variable while in the trade. While out of the trade, the switch will take the value of zero. When the sell signal is found the value of -1 will be assigned. If the stop is hit, the value of -2 will be assigned.

```
trade:= If(PREV <=0, If(bsig, C, 0),
If(ssig, -1, If( H >= PREV+profit, -2, PREV)));
```

Now for the entry signal, look for the formula to have a positive number when it was zero or negative on the previous bar

```
trade > 0 AND Ref(trade,-1) <= 0
```

In a system test the exit signal will be any value less than zero. However, in an expert advisor, individual signals can be set for the sell signal or the stop by looking for trade to be either -1 or -2.

Notice that the way the logic is constructed, a negative value is only held for one bar. On the next bar, the entry signal will be checked and either a new trade started or the value will be set to zero again.

Step 3: The Final Formula

buy signal:

```
profit:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
trade:= If(PREV <=0, If(bsig, C, 0),
If(ssig, -1, If( H >= PREV+profit, -2, PREV)));
trade > 0 AND Ref(trade,-1) <= 0
```

sell signal found:

```
profit:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
trade:= If(PREV <=0, If(bsig, C, 0),
If(ssig, -1, If( H >= PREV+profit, -2, PREV)));
trade = -1
```

profit target hit:

```
profit:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
trade:= If(PREV <=0, If(bsig, C, 0),
If(ssig, -1, If( H >= PREV+profit, -2, PREV)));
trade = -2
```


Trailing stop

A common long position trailing stop is to subtract some multiple of the Average True Range (ATR) from the best price since the trade started. Other stops will use the lowest low of the past few bars. There are many different variations and some new ones will surely be thought up within the next year if not sooner. Regardless of the calculation method, a trailing stop will need to update its value on each bar as the trade progresses.

Exercise 2

The buy and sell signals are the same moving average conditions as the previous exercise. Instead of the profit target stop add a trailing stop set at twice the value of a 14 period ATR from the highest high since the trade started.

Solution

Step 1: An Analysis

This is easiest to do if the logical switch stores the current value of the stop.

Step 2: The Logic

The same variables are assigned at the beginning of the formula. The value of the profit stop has been replaced with the calculation of the stop.

```
tstop:= H - (2 * ATR(14));  
bsig:= Cross( C, Mov(C, 40, S));  
ssig:= Cross( Mov(C, 40, S), C);
```

Now to code the switch; notice when the buy signal is true in the first line the value of the stop is assigned instead of the closing price. When the switch checks if the stop was hit, the low is compared to the value of PREV. Finally if no signals are found, instead of use using PREV the formula takes the highest value of the stop calculation and the previous stop. This way the stop follows the prices as they increase, but maintains its highest value if the price falls.

```
trade:= If(PREV <=0, If(bsig, tstop, 0),  
If(ssig, -1, If( L <= PREV, -2, Max(PREV, tstop))));
```

Step 3: The Final Formula

The final formula is finished the same as the previous formula. The only difference is that a value of -2 now means the trailing stop was hit. Since the formulas are all the same except for the last line, only the buy signal is listed this time.

```
tstop:= H - (2 * ATR(14));  
bsig:= Cross( C, Mov(C, 40, S));
```

```
ssig:= Cross( Mov(C, 40, S), C);
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2, Max(PREV, tstop))));
(trade > 0) AND (Ref(trade,-1) <= 0)
```

Maximum Loss stops

Another common stop is the max loss stop. This simply states no matter what else happens, do not hold the trade if the price falls this much.

Sometimes these stops are based on equity loss. A formula can not do that since it has no way to know the size of the trade or what percentage of equity that represents. Otherwise the logic is very straight forward. Set the stop value and wait for it to be triggered.

Exercise 3

Use the same moving average crossover buy and sell signals. Add a 5 point maximum loss stop. After the price has increased by 5% from the entry price, reset the stop to a break even value of the entry price plus 1%.

Solution

Step 1: An Analysis

This one threw a twist into to the mix. We can not just assign the stop when the buy signal is found. It needs to be changed later when the trade becomes profitable. One way to do this is to get the entry price by adding the stop amount back to the value PREV is carrying.

Step 2: The Logic

Again the variables are defined first. This time the stop has three variables, the initial stop amount, the target percentage and the percentage to reset it to. With this stop, the values can be easily changed, but if the logic is to be changed (say from a percentage to a points calculation), the formula will have to be rewritten instead of just changing the variables.

```
mlstop:= 5;
reset:= 5;
breakevenstop:= 1;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
```

Notice that while the stop still only has two exit conditions, another If() is added to reset the stop at the appropriate time.

```
trade:= If(PREV <=0, If(bsig, C - mlstop, 0),
```

```
If(ssig, -1, If( L <= PREV, -2,
If(H > (PREV+mlstop) * (1+(reset/100)),
(PREV+mlstop) * (1+(breakeven/100)), PREV))));
```

Step 3: The Final Formula

The final formula is then finished as the previous stops formulas were

```
mlstop:= 5;
reset:= 5;
breakeven:= 1;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
trade:= If(PREV <=0, If(bsig, C - mlstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H > (PREV+mlstop) * (1+(reset/100)),
(PREV+mlstop) * (1+(breakeven/100)), PREV))));
trade > 0 AND Ref(trade,-1) <= 0
```

Time based stops

This type of stop is aimed at directing investments where they will be the most profitable. It will close a trade after a certain amount of time. This stop is normally combined with other stops but will be introduced by itself

Exercise 4

Use the same buy and sell conditions as before. Add a stop to close the trade after 10 bars if the sell signal has not been triggered.

Solution

Step 1: An Analysis

This is a bit tricky. It could be done by setting the switch to the number 1 when the trade starts. This value could be increments on each successive bar until the target number was reached. However, there is another way.

Step 2: The Logic

Start by defining the variables:

```
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
```

When the trade starts, the switch will still take the value of 1. However, it will not increment this value. Instead BarsSince() will be used to get the number of bars that have past since PREV was less than or equal to zero.

```
trade:= If(PREV <=0, If(bsig, 1, 0),  
If(ssig, -1, If( BarsSince(PREV<=0)>=time, -2, PREV)));
```

Step 3: The Final Formula

The final formula is finished as before.

```
time:= 10;  
bsig:= Cross( C, Mov(C, 40, S));  
ssig:= Cross( Mov(C, 40, S), C);  
trade:= If(PREV <=0, If(bsig, 1, 0),  
If(ssig, -1, If( BarsSince(PREV<=0)>=time, -2, PREV)));  
trade > 0 AND Ref(trade,-1) <= 0
```

A similar method could have been used when resetting the max loss stop. ValueWhen() could have looked back to the start of the trade and gotten the value of the close on that bar.

Putting it all together

Each of these stops are fairly easy to code once the proper switch logic is explained. However, stops are rarely used just one at a time.

Exercise 5

Use the same buy and sell signals as before. Add to this the profit target stop, the trailing stop, and time stop from the previous exercises.

Solution

Step 1: An Analysis

The switch will need to track the current value of the trailing stop as this is value cannot easily be calculated otherwise. Use ValueWhen to get the entry price for the profit target. Use BarsSince for the time stop.

Step 2: The Logic

Once again start by defining the variables

```
profit:= 10;  
tstop:= H - (2 * ATR(14));  
time:= 10;
```

```
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
```

There are three different stops so the switch will add a -3 and -4 signal values. It does not really matter which stop has which value. In cases like this a comment statement is worth while for future reference.

```
{ trade exit values
-1 = sell signal
-2 = trailing stop
-3 = profit target stop
-4 = time stop
}
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
```

Step 3: The Final Formula

Here is the final formula. The checks at the end are still the same except an expert can now check for four different exit signals.

```
profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
{ trade exit values
-1 = sell signal
-2 = trailing stop
-3 = profit target stop
-4 = time stop
}
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
trade > 0 AND Ref(trade,-1) <= 0
```

It is worth noting how many PREVs and nested Ifs are in this last formula. It is possible to use so many that MetaStock is not able to preform the calculation. MetaStock would not return anything. The entire formula would be undefined.

The exact point this happens is not fixed. The above formula could possibly be extended another condition or two or it could already be at the limit. Usually I start worrying after putting 5 or 6 PREVs in one formula. I have also gone about 7 levels deep on a nested If. When the two are combined, the complexity of the formula increases faster so more discretion is required.

In general, if a formula does not plot anything and you do not see a logic error, try simplifying it. Reduce the number of Prevs or restructure the If logic so it is not as deep.

Chapter 10: Extrapolating Higher Time Frames

Many systems and patterns look for events to occur in more than one time frame. For example, a signal may require an indicator to be oversold in both daily and weekly times frames. This is not easily done in MetaStock. It is usually possible, but requires the longer term signals be constructed manually.

The problem is caused by the MetaStock Formula Language not having any functions for time frames. There is no Weekly() function to wrap around a moving average to get a weekly version of that moving average. Instead, the formula must identify the weekly values, sum them, and then divide the sum by the time period of the average.

Starting Simple

Chapter 7, Pivot Points, explained how to get a daily High, Low, or Closing price from intraday data. This same logic can be used for weekly or monthly values with just a few minor changes.

Exercise 1

Write a formula for a weekly 10-period simple moving average to be calculated on daily data. Calculate the average on the closing price. The formula can not use Ref() with a positive number of periods or the average will not have a value on the last few bars in the chart.

Solution

Step 1: An Analysis

As shown in the pivot point chapter, you can set a variable to identify the start of a new day by looking for a change in the value of the DayofMonth() function. The DayofWeek() function can be used to find the start of week with just a slightly different coding. The most recent 10 weekly closes then have to be summed before they can be averaged.

Step 2: The Logic

The DayofWeek() function numbers the days of the week from 1 to 7. When it changes to a lower number, a new week has started. Therefore, the condition for the start of a new week would be:

```
new:=ROC(DayOfWeek(),1,$)<0;
```

The logic for the close of the of the last bar of the previous week would be the same as in the pivot formula:

```
yc:=ValueWhen(1,new, Ref(C,-1));
```

The formula is just looking back to the most recent time the new variable was true. By changing the definition of new, the formula can be set to extrapolate different time intervals with little to no other changes required.

The last ten weekly closes can be summed with ten ValueWhen() functions:

```
valuewhen(1, new, Ref(c,-1)) +  
valuewhen(2, new, Ref(c,-1)) +  
valuewhen(3, new, Ref(c,-1)) +  
valuewhen(4, new, Ref(c,-1)) +  
valuewhen(5, new, Ref(c,-1)) +  
valuewhen(6, new, Ref(c,-1)) +  
valuewhen(7, new, Ref(c,-1)) +  
valuewhen(8, new, Ref(c,-1)) +  
valuewhen(9, new, Ref(c,-1)) +  
valuewhen(10, new, Ref(c,-1))
```

However, this does not scale well for longer averages. It is impractical, especially when there is an alternative. First, assign the number of periods for the average to a variable. Changing that variable will change the average's length. Then, instead of using ValueWhen() to get the last close, setup an If() that will take the value of the previous close when new is true and at all other times will have the value of zero:

```
pds:= 10  
total:= cum( if( new, Ref(C, -1), 0) );
```

Cum() was used to add all the weekly closes together and assign that to the variable total. Now, by subtracting the value of total some time in the past from the current total, we will get the sum of the weekly closes between that past time and now. That difference is then divided by the number of periods for the average:

```
(total - valuewhen(pds+1, new, total))/pds
```

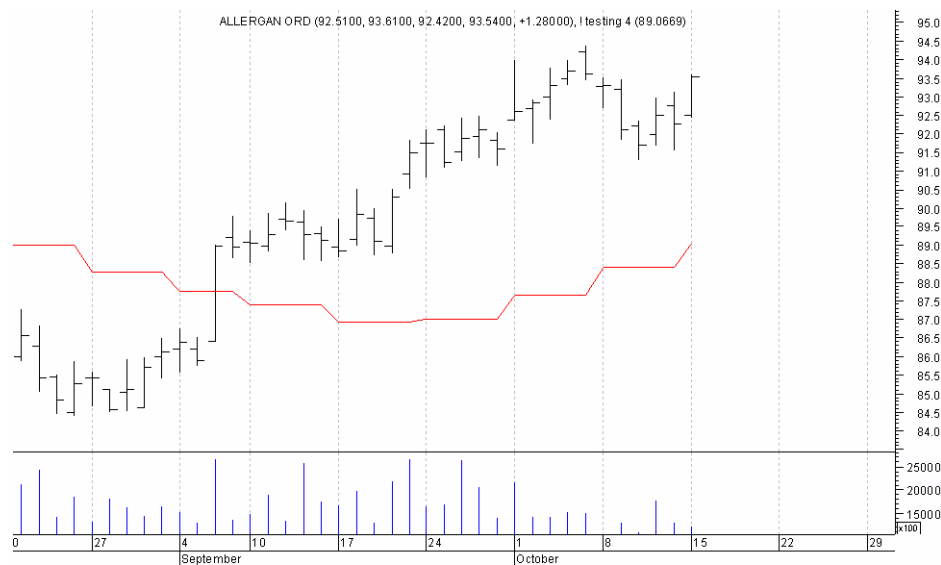
One was added to the pds variable to get the value of total prior to the last ten values. If pds was used by itself, the total of the closing prices would be one value short.

Step 3: The Final Formula

Putting all the logic together, the final formula would look similar to the code below;

```
pds:= 10;  
new:=ROC( DayOfWeek(), 1, $ )<0;  
total:= Cum( If( new, Ref(C, -1), 0) );  
(total - ValueWhen(pds+1, new, total))/pds
```


Applied to a daily chart, it would look like this:



If a time frame other than weekly is desired, only the variable “new” needs to be changed. As written, it is looking for the rate of change of the day of week function to be negative. A daily average can be extrapolated from intraday data by looking for any change in the day of week function:

```
new:=ROC (DayOfWeek ( ) , 1 , $) <> 0 ;
```

Since all bars of a given day will be for the same day of week, when a new day of week is found a new day must have started. In this case, the DayOfMonth function could also be used with the same logic:

```
new:=ROC (DayOfMonth ( ) , 1 , $) <> 0 ;
```

Similarly, a monthly value can be extrapolated by looking for any change in the Month function:

```
new:=ROC (Month ( ) , 1 , $) <> 0 ;
```

Should the Month function decrease in value, it would signal the start of a new year. A new year could also be looked for by looking for any change in the Year() function:

```
new:=ROC (year ( ) , 1 , $) <> 0 ;
```

Whatever time frame is being extrapolated, the formula will need sufficient data for the calculation. However, using intraday data to calculate a yearly average is not usually possible as it would require several years of intraday data be loaded in the chart.

Exponential Changes

The simple moving average was just a summation and a division. Other types of moving averages require more involved calculations. Consider the exponential moving average (EMA). It is calculated by multiplying the value of the current bar by some value alpha. The value of the average from the previous bar is multiplied by one minus the same alpha. The results of these two calculations are added together to get the current value of the average. If the average was of the Close, the formula could be expressed this way:

$$\text{Current EMA} = (C * \alpha) + (\text{previous EMA} * (1 - \alpha))$$

The value of alpha is equal to two divided by the number of periods in the average plus one:

$$\alpha = 2 / (\text{EMA time periods} + 1)$$

Exercise 2

Write a formula to extrapolate an hourly EMA of the Close from 5 minute data. The formula should seed the EMA calculation with the Close of the first hour of the chart. Use 40 time periods but set this value in a variable so it can be easily changed later.

Solution:

Step 1: An Analysis

Since each value of an EMA depends on the previous value of the average, a summation of the values to be averaged will not help. Instead, the formula will have to calculate the value each time a new time interval for the extrapolated time period starts. Between those time intervals, the last calculated value will need to be remembered so it can be used to derive the next value in the average. This will require the use of the PREV function

Step 2: The Logic

Start with the initial values that will not change. The time interval is fixed but assigned to a variable. Alpha is also fixed but still needs to be calculated:

```
tp:= 40; {EMA time periods}  
alpha:= 2 / (tp +1);
```

Next, the formula to identify the start of a new time period should be done. This is best near the top so it is easily changed allowing the basic logic of a formula to be reused with just minor

changes. The formula for an hourly extrapolation will look for a change in the value of the Hour() function. Since the 5 minute interval is where the formula will be applied and 5 minutes is evenly divisible into 60 minutes, this extrapolation is possible. The same logic for a daily extrapolation can be used. Just change the DayOfWeek() function to the Hour() function:

```
new:=ROC(Hour(),1,$)<>0;
```

This identifies when a new hour starts. However, the average needs to use the Close of the previous hour for the calculation. So, each time new is true, the Ref() function is used to get the previous bar's close for the new value in the average. If new is not true, use PREV to "remember" the last value of the average. This same PREV is used for the value of the average from the previous interval to calculate the next of the average:

```
EMA:= If( new, ( Ref(C, -1) * alpha) + ( PREV * (1-alpha)), PREV);
```

However, this calculation does not seed the average with the close of the first hour in the chart. PREV has the value of zero at the start of the chart. This means the value of EMA will start at zero and slowly increase over the next several hours before it gets to a value close to what the prices are. Typically, this would take one to two times the length of the average. To avoid that, the average can be assigned a value to start with. This seed value will then be used by PREV as a jump-start, instead of the value of zero.

To seed the average, another If() must be combined with the first. When new is true, this If() is checked first and can only be true on the first hour of the chart. The Cum() function can be used to count the number of times new has been TRUE:

```
EMA:= If( new, If( Cum( new ) = 1, Ref(C, -1),  
( Ref(C, -1) * alpha) + ( PREV * (1-alpha))), PREV);
```

Step 3: The Final Formula

Now all that remains is to call the EMA variable so the formula will plot the value. The final formula should look something like this:

```
tp:= 40; {EMA time periods}  
new:=ROC(Hour(),1,$)<>0;  
alpha:= 2 / (tp +1);  
EMA:= If( new, If( Cum( new ) = 1, Ref(C, -1),  
( Ref(C, -1) * alpha) + ( PREV * (1-alpha)), PREV));  
EMA
```

The new and alpha variables were switched to put new closer to the top of the formula. This is only to make that part easier to locate so reuse and changes are faster and easier to make.

Relative Applications

The two moving averages just shown are fairly straightforward in extrapolating, but they demonstrate the two types of calculations required. The simple moving average required identifying and referencing several specific points in time to get the data from that point for the calculation. The exponential moving average required keeping a current value that was adjusted only at specific points in time. These two methods are used individually or in combination for all extrapolations.

Exercise 3

The formula for the RSI was explained Chapter 3 (page 13). Extrapolate this formula to calculate a weekly RSI using daily data. Write the formula as an indicator that will prompt for the number of time periods for the RSI.

Solution

Step 1: An Analysis

For convenience, the RSI formula is reprinted here:

```
change:= ROC(C,1,$);  
Z:=Wilders(If(change>0,change,0),14);  
Y:=Wilders(If(change<0,Abs(change),0),14);  
RS:=Z/Y;  
100-(100/(1+RS))
```

This version uses the Wilders smoothing function. It will need to be replaced with the longer version of the calculation. Again copying from Chapter 3 (page 13), that formula is:

```
If( cum(1) = 14, Sum(C, 14)/14,  
PREV + ( (C-PREV)/14) )
```

The formulas above are written to use 14 periods for the calculations. This hard-coded notation will need to be replaced with a variable and an Input function used to assign the variable.

Step 2: The Logic

The variable for the time periods with the input prompt is easily done. Inputs are best grouped at the top so they are quickly found and can be edited or replaced as needed.

```
tp:= Input("time periods for RSI",2,200,14);
```

This formula is another weekly extrapolation from daily data so the new variable is the same as the one used in the simple moving average:

```
new:=ROC(DayOfWeek(),1,$)<0;
```

The change variable in the original RSI calculated the point difference of the close from the previous bar. The current formula needs to get the difference between the last price of the most recent week and the week before that. ValueWhen() can get those values like it did in the simple moving average extrapolation. By subtracting the two ValueWhen() functions, we get the effect of the rate of change function:

```
change:= ValueWhen(1,new,Ref( C,-1))-ValueWhen(2,new,Ref(C,-1));
```

The next two lines of the original RSI nest an If() function inside a Wilders() function. For easier readability, this logic will extract the If() and do it first. The Z variable was defined as the smoothing of the positive change. Here, Z1 will be defined as having a non-zero value only when a new week starts and change is greater than zero:

```
Z1:= If( new AND change>0, change, 0);
```

Z can now be defined as a Wilders smoothing of Z1. However, Z must still be extrapolated to be a weekly calculation. Wilder's smoothing requires the first calculation be a simple moving average. The Cum() of new counts the number of weeks, but the Sum() used in the daily version must be replaced by Cum() for the first average. In English:

If the total number of the start of new weeks equals the inputted number of time periods then total all values of the variable Z1 from the start of the chart to the current bar and divide that by the number of time periods, tp.

As a formula, this is written:

```
Z:=If(Cum(new)=tp,Cum(Z1)/tp, {second half} )
```

The second half of the formula Wilders smoothing is a mathematical calculation applied to the last value it had. This is only done on the start of a new week so a second If() is nested to see if that occurred. If it is the start of a new week, the previous value of the average is subtracted from the new value of change. The result is divided by the number of time periods and then added to the previous value of the average. If this is not the start of a new week, the last calculated value needs to be remembered:

```
If(new, (PREV+ ((Z1-PREV)/tp), PREV);
```

This calculation uses the PREV function three times. Logically, the formula is good, but each use of PREV adds to the memory usage and calculation time of MetaStock formulas. This is where Algebra helps. The math can be rearranged to remove one of the PREVs. It is beyond the scope of this text to explain but:

$PREV + ((Z1 - PREV) / tp)$ is equal to $PREV * (tp - 1) + Z1 / tp$

This changes the second half of the Z variable to:

```
If (new, (PREV* (tp-1) +Z1) /tp, PREV)
```

Putting the Z1 and completed Z variables together, the formula now shows:

```
Z1:= If ( new AND change>0, change, 0) ;  
Z:=If (Cum(new)=tp, Cum(Z1) /tp, If (new, (PREV* (tp-1) +Z1) /tp, PREV) )
```

The Y variable is treated the same way. Y1 is created for the nested If(). The logic for the extrapolated Y variable is exactly the same as for Z except it references Y1 instead of Z1:

```
Y1:=If (new AND change<0, Abs (change) , 0) ;  
Y:=If (Cum(new)=tp, Cum(Y1) /tp, If (new, (PREV* (tp-1) +Y1) /tp, PREV) ) ;
```

The formula given for the RSI has a flaw; it is possible to get “Divide by zero” errors as it is written. The custom version of the RSI from Chapter 3 (page 15) added code to prevent this. This exercise did not ask anything be done about the error but the fix is easy to include. The next step in the formula divides Z by Y. However, if the prices have been in an extended uptrend, Y could have a value of zero.

To avoid the error, this formula will use a two step process. First, the divisor, Y, will be assigned to a temp variable. Temp will be assigned the value of -1 if Y is equal to zero. Otherwise, temp will have the save value Y does. Since Y is calculated with only positive values, the -1 in temp is a flag for the second calculation.

```
temp:= If (Y=0, -1, Y) ;
```

Part two does the division after checking for the flag. If the flag is set, a default result of 1000 is returned. 1000 was chosen because it will push the final result of the RS to the near maximum value it should have for prolonged uptrends. This is different than the method used in Chapter 3 where an alternate denominator was used. Both methods work but one may fit a formula’s needs better than the other.

```
RS:= If (temp=-1, 1000, Z/temp) ;
```

The final step of the calculation is not dependent on time or events. It can be done as listed. However, the alterations of this formula to extrapolate weekly values from daily data have had an unintended consequence. The Wilder() function has no value till the required number of time periods have passed. However, the Cum() and ValueWhen() functions become defined much sooner. This formula will plot incomplete and inaccurate values until the specified number of weeks have passed. Then it will be correct till the end of the chart. To avoid the incorrect early values, BarsSince() has been added to the last line. Its only purpose is to make the formula undefined till the calculation will be accurate.

The BarsSince is nested inside an If(). If BarsSince is greater than or equal to zero, i.e. the required data is available, the final calculation is made. BarsSince will always be zero or greater if

it is defined, so the zero in the false result for the If() is just to fulfill the syntax requirements of the If() function.

```
If (BarsSince (Cum (new) =tp) >=0, 100 - (100 / (1+RS) ) , 0)
```

Step 3: The Final Formula

Putting all the previous lines together, the final formula would be:

```
tp:= Input("time periods for RSI",2,200,14);
new:=ROC(DayOfWeek(),1,$)<0;
change:= ValueWhen(1,new,Ref(C,-1))-ValueWhen(2,new,Ref(C,-1));
Z1:=If(new AND change>0, change, 0);
Z:=If(Cum(new)=tp,Cum(Z1)/tp,If(new,(PREV*(tp-1)+Z1)/tp,PREV));
Y1:=If(new AND change<0, Abs(change), 0);
Y:=If(Cum(new)=tp,Cum(Y1)/tp,If(new,(PREV*(tp-1)+Y1)/tp,PREV));
temp:= If(Y=0,-1,Y);
RS:= If(temp=-1,1000,Z/temp);
If (BarsSince (Cum (new) =tp) >=0, 100 - (100 / (1+RS) ) , 0)
```

This formula has required more explanation than most. The original calculation was longer and extrapolation required taking extra steps that previously were glossed over or unnecessary. Any extrapolation must look for such things and address them. Failure to do so could result in errors or inaccurate results.

Stochastics

The Stochastic Oscillator is another common indicator. It is used in many systems either as part of the main signal, a trend indicator, or for confirmation of another signal. This increases the likelihood of needing to extrapolate its value.

Exercise 4

Write a formula for a standard 5 / 3 / 3 Stochastic Oscillator crossing below its signal line. Both lines must be greater than 80 on the bar of the cross. The stochastic is to be calculated on monthly data but is to be plotted on a daily chart.

Solution

This formula will be long, but can be done in stages. The Stochastic Oscillator's formula can be expressed this way:

```
sto:= ( Sum( C - LLV( L, 5 ), 3 ) /
Sum( HHV( H, 5 ) - LLV( L, 5 ), 3 ) ) * 100;
sig:= Mov( sto, 3, S );
```

```
sto;  
sig;
```

Sto is the main line of the stochastic. The 5 is the %k value and the 3 is the %k slowing. These are the first two numbers in the specification of the stochastic's time periods. The signal line is just a moving average of the main stochastic. The 3 used there is also called the %d value. With the above formula, the extrapolation can be done as follows:

Step 1: An Analysis

The exercise specifies as 5 / 3 / 3 stochastic but it is easy to set those as variables. Like the formula for the daily stochastic and signal line, this formula should assign those calculations to variables. Then the variables can be used to build the signal condition.

Step 2: The Logic

Constant variables should always be defined at the top. In this case, the variable names are longer than normal but this makes their values easier to understand. Comments can also be added if desired:

```
kper:=5;    {%k periods}  
ksper:=3;   {%k slowing periods}  
dper:=3;    {%d periods}
```

This formula is to extrapolate a monthly indicator. New should therefore look for any change in the value of Month():

```
new:=ROC (Month ( ) , 1 , $) <> 0 ;
```

The stochastic uses the lowest value of the low and the highest value of the high calculated over %k time periods. HHV() and LLV() require a set number of periods. Even though an average month will have 22 trading days, holidays and special events alter that. To be accurate, HighestSince() and LowestSince() will be used instead. The %k periods becomes the number of instances to look back to during the calculation. If the calculation is assumed to be from the last day of the month, the following lines will accurately calculate the highest high and lowest low over the desired number of months:

```
ll:=LowestSince (kper , new , L) ;  
hh:=HighestSince (kper , new , H) ;
```

The smoothing of the Stochastic Oscillator is done by summing the numerator and denominator values. Since both are summed over the same number of periods, no division is needed as the ratio is preserved without it. Sum() can not be used with an unknown number of periods so Cum() is the only alternative.

Just as with the moving average formula, a cumulation will be made of all the data and an earlier value of the cumulation subtracted from the current one. This will give the sum of just the most recent values.

Two cumulations are needed, one for the numerator and a second for the denominator. The numerator totals the close minus the lowest low value. The denominator totals the highest high value minus the lowest low value.

```
c1:=Cum(If(new,Ref(C-l1,-1),0));
c2:=Cum(If(new,Ref(hh-l1,-1),0));
```

Since the formula is nowhere near the 20 variable limit, the next section of logic will use several variables to finish calculating the numerator and denominator of the stochastic. A current value of the cumulations will be assigned to variables cn and cd:

```
cn:=ValueWhen(1,new,c1);
cd:=ValueWhen(1,new,c2);
```

ValueWhen() was used so the calculation will remain the same through the month till a new month starts. This method does not require PREV or an If() allowing the formula to calculate faster. ValueWhen() functions will also be used for the past values of the numerator and denominator. However, since the most recent instance is the current value, one must be added to the %k slowing time periods to get the value of the cumulation before the summation for the current data.

Consider; if this current calculation is for June and the formula is using a 3 period %k slowing, the calculation is to include the data for April, May and June. ValueWhen(1, new, c1) returned the value of c1 for the month of June. This is the cumulative values of c1 through all the loaded data. ValueWhen(3, new, c1) would be the value of c1 for the month of April. If this was subtracted from the one for June, the result would just be the summation of May and June data. The formula has to get the value before April (March) to get the correct summation:

```
pn:=ValueWhen(ksper+1,new,c1);
pd:=ValueWhen(ksper+1,new,c2);
```

The denominator could take the value of zero. On monthly data, this is not very likely, but since new could be changed later to refer to hourly data, it is better to include a divide by zero trap now. Once again, this will be done with an extra variable. The denominator will normally be the result of cd - pd. If this value is 0, the denom variable will set a flag by taking the value -1.

```
denom:=If(cd-pd=0,-1,cd-pd);
```

Finally, the main line of the stochastic can be calculated. If the flag is set, a default value needs to be chosen. Since the stochastic measures the position of the price between the range of the data, a zero range means the price is simultaneously at the high and low of range. This is a confused signal so the formula will default to a value of 50. Otherwise, the result of the cn - pn will be divided by the denominator and that result multiplied by 100.

```
mstok:=If(denom=-1,50,((cn-pn)/denom)*100);
```

Now a third cumulation of the monthly stochastic is needed for the signal line. The signal line is just a simple moving average. It uses the same logic explained in Exercise 1, page 74. Please review that section if the logic below is not clear.

```
c3:=Cum(If(new,mstok,0));  
mstod:=(c3-ValueWhen(dper+1,new,c3))/dper;
```

Now mstok and mstod define the main and signal lines of the Stochastic Oscillator. It is now possible to construct the signal. To review, the signal conditions were:

1. The main stochastic line is crossing below the signal line
2. The main stochastic line is greater than 80
3. The signal line is greater than 80

Since both lines must be greater than 80, by checking if the smallest is greater than 80, the largest must also be greater than 80:

```
Min( mstok, mstod ) > 80
```

This just leaves the cross. That is easy to do. ValueWhen() could be used to compare the previous month's values to the current, but it is not necessary. The Cross() function says the first value is greater than the second on the current bar but was less than the second on the previous bar. Even though this is an extrapolation of monthly data, it is being done on daily values. The change in value will only occur at the start of a new month. The last value of one month and the first value of the next month shows the same comparison as first values of the last two months. Thus

```
Cross( mstod, mstok )
```

is the same as:

```
ValueWhen(1, new, mstod) > ValueWhen(1, new, mstok) AND  
ValueWhen(1, new, mstod) < ValueWhen(1, new, mstok)
```

Since the Cross() is much simpler, that will be used.

Step 3: The Final Formula

Putting all that together, the final formula will be:

```
kper:=5;    {%k periods}  
ksper:=3;   {%k slowing periods}  
dper:=3;    {%d periods}  
new:=ROC(Month(),1,$)<>0;  
ll:=LowestSince(kper,new,L);
```

```

hh:=HighestSince(kper,new,H);
c1:=Cum(If(new,Ref(C-11,-1),0));
c2:=Cum(If(new,Ref(hh-11,-1),0));
cn:=ValueWhen(1,new,c1);
cd:=ValueWhen(1,new,c2);
pn:=ValueWhen(ksper+1,new,c1);
pd:=ValueWhen(ksper+1,new,c2);
denom:=If(cd-pd=0,-1,cd-pd);
mstok:=If(denom=-1,50,((cn-pn)/denom)*100);
c3:=Cum(If(new,mstok,0));
mstod:=(c3-ValueWhen(dper+1,new,c3))/dper;
Min(mstok,mstod) > 80 AND Cross(mstod,mstok)

```

This chapter has gone into much more involved logic than the previous chapters. It has taken what was given before and applied it in ways that may not have been previously thought of. If something is not clear, please review that section again. This text is almost done, but the most complex formula logic was discussed in these last two chapters.

Chapter 11: Value to Generate Signal

Sometimes it is important to know what must happen for a signal to occur. Consider, a chart where the price is getting close to a moving average but has not crossed it yet. Knowing what the price must be in order to cross the moving average allows the trader to place a stop order anticipating the signal.

A Simple Example

This sort of formula is usually very easy to write after a much more complicated process of setting up the equation. For these formulas, the signal must be written out as a mathematical equation. Then it is solved for the price. An example of this is the simple moving average crossover.

Exercise 1

Write a formula to display the closing price required on the next bar for the close to cross its 10 period simple moving average

Solution

These types of formulas are all about setting up the equation. Algebra is more important here than computer knowledge.

Step 1: An Analysis

This formula assumes the Close is currently above or below the 10 period moving average. It does not matter which one. All the formula is seeking is the price at which the moving average and the close will be equal to each other. If the next bar's close is that price or further from the current current close, a cross will occur.

Step 2: The Logic

To setup this formula, the moving average must be written out as a mathematical equation. For the understanding of the widest possible audience, this text will use only the most basic mathematical notation. As such, a 10 period simple moving average could be written as:

$$\begin{aligned} & (C + \text{Ref}(C, -1) + \\ & \text{Ref}(C, -2) + \text{Ref}(C, -3) + \\ & \text{Ref}(C, -4) + \text{Ref}(C, -5) + \\ & \text{Ref}(C, -6) + \text{Ref}(C, -7) + \\ & \text{Ref}(C, -8) + \text{Ref}(C, -9)) / 10 \end{aligned}$$

To get the formula for the value of tomorrow's moving average, all the Ref() function are incremented by 1:

$$\begin{aligned} & (\text{Ref}(C, 1) + C + \\ & \text{Ref}(C, -1) + \text{Ref}(C, -2) + \\ & \text{Ref}(C, -3) + \text{Ref}(C, -4) + \\ & \text{Ref}(C, -5) + \text{Ref}(C, -6) + \\ & \text{Ref}(C, -7) + \text{Ref}(C, -8)) / 10 \end{aligned}$$

Notice the first value of the close (on the first line), now uses Ref() with +1 to get the value of the close on the next bar.

Using the Sum function, we could shorten that formula to:

$$(\text{Ref}(C, 1) + \text{Sum}(C, 9)) / 10$$

To trigger a cross, the close of the next bar must equal the above formula:

$$\text{Ref}(C, 1) = (\text{Ref}(C, 1) + \text{Sum}(C, 9)) / 10$$

Now the equation is solved for the value of Ref(C, 1). Start with the equation:

$$\text{Ref}(C, 1) = (\text{Ref}(C, 1) + \text{Sum}(C, 9)) / 10$$

Multiply both sides by 10:

$$\text{Ref}(C, 1) * 10 = \text{Ref}(C, 1) + \text{Sum}(C, 9)$$

Subtract the Ref(C, 1) from the both sides to move it from the right side of the equation to the left side

$$(\text{Ref}(C, 1) * 10) - \text{Ref}(C, 1) = \text{Sum}(C, 9)$$

A value multiplied by 10 minus the original value is equal to the original value multiplied by 9.

$$\text{Ref}(C, 1) * 9 = \text{Sum}(C, 9)$$

Divide both sides by 9 to get Ref(C, 1) by itself on the left side:

$$\text{Ref}(C, 1) = \text{Sum}(C, 9) / 9$$

Thus, to get the value needed for the close to cross it's 10 period simple moving average on the next bar, calculate a 9 period simple moving average of the close on the current bar

Step 3: The Final Formula

The formula for a 9 period simple moving average is:

$\text{MOV}(C, 9, S)$

As was stated previously, the derivation of the logic and math for this formula was far more involved than the actual formula. A second example will emphasize this even more.

More Math

Another commonly used indicator is the exponential moving average. It is used to smooth numerous functions and is the basic component of the MACD. This example will again keep things simple and focus on a close crossing its own moving average.

Exercise 2

Calculate what the close must be on the next bar to intersect a 40 period exponential moving average of the close.

Solution

The same procedure is used. Setup the mathematical equation and solve for the value, in this case the close of the next bar.

Important: The exponential moving average is usually seeded (or started) with a value at the beginning of the data. Without this seed, the formula takes about 2-3 times the average length to be usable. This is because PREV will start at zero on the first bar and it takes that long for the formula to reach the values it is averaging. To keep this example simple, we will assume there are hundreds of prior bars of data loaded. Therefore, the formula can be written without a seed value.

Step 1: An Analysis

The MetaStock formula for an exponential moving average is

$(C * \alpha) + (PREV * (1 - \alpha))$

where alpha is equal to 2 divided by 1 plus the number of periods in the moving average.

Step 2: The Logic

To reduce the math requirements, alpha will be kept as a variable.

PREV is equal to the value of the moving average on the previous bar. When the formula is applied to the next bar, PREV becomes equal to the current value of the moving average. Thus, the equation for the moving average on the next bar is:

$$(\text{Ref}(C, 1) * \alpha) + (\text{Mov}(C, 40, E) * (1 - \alpha))$$

To get the value the close must have on that bar to intersect the moving average, set the average to equal the close and then solve for the value of the close.

$$\text{Ref}(C, 1) = (\text{Ref}(C, 1) * \alpha) + (\text{Mov}(C, 40, E) * (1 - \alpha))$$

The equation may look complicated, but in this case, it solves to another really simple formula. First, all references to $\text{Ref}(C, 1)$ should be on the same side of the equation. To do this, $\text{Ref}(C, 1)$ times α is subtracted from both sides:

$$\text{Ref}(C, 1) - (\text{Ref}(C, 1) * \alpha) = (\text{Mov}(C, 40, E) * (1 - \alpha))$$

Now look at the left side of the equation:

$$\text{Ref}(C, 1) - (\text{Ref}(C, 1) * \alpha)$$

If X is substituted for the close of the next bar, the formula could be written as:

$$X - (X * \alpha)$$

Algebraically, this is the same thing as:

$$X * (1 - \alpha)$$

So, that same rule is applied to the left side of the formula:

$$\text{Ref}(C, 1) * (1 - \alpha) = \text{Mov}(C, 40, E) * (1 - \alpha)$$

notice how both sides are now being multiplied by the same quantity. to finish solving, divide both sides by $(1 - \alpha)$

$$\text{Ref}(C, 1) = \text{Mov}(C, 40, E)$$

So according to the math, the value of the close needed on the next bar to intersect a 40 period EMA of the close is the current value of the EMA.

Step 3: The Final Formula

As was promised, a simple solution:

$$\text{Mov}(C, 40, E)$$

All is not so Simple

Please do not think all such formulas are so simply resolved. This text intentionally chose these examples because the math was easier to work through. Some indicators require advanced mathematics to solve. It is beyond the scope of this text to teach the math skills required to derive the calculation needed for those formulas.

Once the calculation is known, it is usually possible to write a simple MetaStock formula to express that calculation. However, this may not always be the case. Some indicators use more than the closing price in their calculation. For example, the stochastic oscillator uses the highest high and lowest low of the past few bars. When calculating what the close must be to trigger a signal, the formula must assume the range stays the same or, if close is outside that range, the close is equal to the new high or low. To do otherwise will require solving for three variables at the same time. This is possible, but the math becomes even more complicated.

When writing these types of formulas, it may reach the point that the MetaStock formula language is not sufficient to the math required. If the solution require multiple iterations, simultaneous equations, or other complex mathematics, a DLL will need to be written. This text is not intended to teach DLL writing. If this is required and the reader is not fluent in computer programming, please seek someone who has those skills for this work.

Metastock does not maintain a list of programmers willing to do this work. There are some that frequent our online forum. These people are under no obligations and any assistance they offer is solely up to them. MetaStock makes no claim or warranty to their service. The forum is provided so MetaStock users may assist each other and share ideas. If the reader wishes to participate in these discussions, the forum URL is below:

<http://forum.metastock.com/Discussions/>

An alternative to the forum is a local college or technical school's computer science department. An agreement could be made with one of the students to write the DLL for a modest fee. This should be less expensive than hiring a professional programmer and will let the student gain work experience.

Regardless who is contacted to do the programming, a copy of the MetaStock Developer's Kit (MDK) will be needed. Currently this is only available from the MetaStock sales department. Please contact them for price and availability.

Chapter 12: Commentaries

Commentaries have a reputation of being hard to write. Some people are intimidated by the prospect and do not even try. They might think of commentaries and immediately decide it is not worth effort. The truth is, after everything else in this text, commentaries are easy. As proof, this chapter will walk through the process of creating a commentary.

What does a commentary do?

Commentaries are designed to provide analysis of data in a chart. This analysis is a combination of several different elements:

- Fixed text
- Calculated values and data
- Conditional text
- Links to external sources

The fixed text is information that will always be displayed. This could be a disclaimer at the bottom of the commentary. It could include an explanation of a trading system or notes about the author. It also includes labels and connecting information for the other elements.

Calculated values and data are used to display things like the close of the current bar, or the value of a moving average. It could also include the date and time of the current bar. There are even data values for the name and symbol of the security the expert is applied to.

Conditional text is what gives the commentary its usefulness. WriteIf() functions can check various conditions and display text based on the result. This is the part that requires a bit of forethought, but otherwise, is no more complicated than using If() functions in indicators.

Links to external sources are the final element that can be included. This could be an imbedded video or audio clip. It could also be a link to an HTML file or web site. This is not as commonly used, but may be needed for some projects.

The commentary editor has some features similar to a word processor. It allows text to be centered or adjusted to either margin. Text can be displayed in different fonts and/or colors. It can be underlined, bolded, or italicized. Depending on the extent of the text formatting desired, it may be easier to write the text and format it in a program like MicroSoft Word and then copy it into the commentary.

What is needed to write a commentary?

As was mentioned earlier, a commentary requires a bit of forethought. Like short indicators, a short commentary could be written off the cuff. However, the longer ones should be given some

thought ahead of time. Before starting the actual commentary code, the following should be prepared:

- the system the commentary will be use for
- a list of all values the commentary should show
- a list of each condition the commentary will provide analysis on
- at least a basic outline of the text to be displayed for each condition

For the sample commentary, this text will refer back to the final formula in the Custom Stops chapter: “Putting it all together” on page 71. This is the formula for the enter long signal:

```
profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
{ trade exit values
  -1 = sell signal
  -2 = trailing stop
  -3 = profit target stop
  -4 = time stop }
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
trade > 0 AND Ref(trade,-1) <= 0
```

To review, this system will enter a long trade when the close crosses above its 40-period simple moving average. It will exit when any of the following happens:

- the close falls below its 40-period simple moving average
- the trade has been active for 10 bars
- the trade has made a profit of 10 points or more
- a trailing stop is hit.

A good commentary should have conditional text for all four exits and the entry signal. This commentary will also show the following:

- the name of the expert
- the name and symbol of the security the expert is attached to
- the date of the current bar
- the current value of the close and the moving average
- how long till the time stop is triggered
- the profit target for the trade
- the current value of the trailing stop

For a layout, the Commentary will display the names, symbol, and date at the top. This will be Centered with the names in a larger text to emphasize them. The current values of the closing price and the moving average will be listed on the left edge. Below that will be additional text for any active trade.

The actual text used to describe the reason for the trade signals will be marked by comments in this example. In actual use, this can be replaced with any desired message but for the purpose of this exercise, showing where the text would go is enough.

The Initial Framework

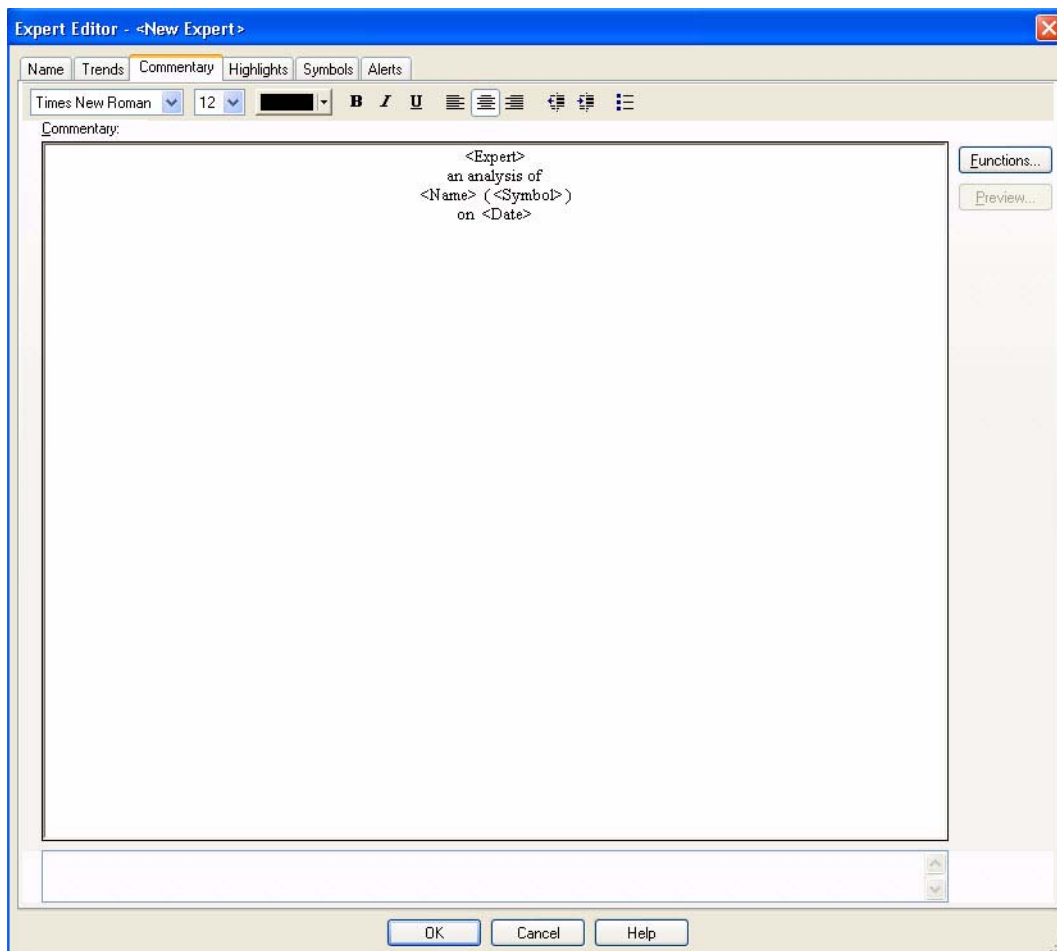
Much of a commentary will not change. This can usually be done first to give a framework to put the rest of the commentary in. For this example, the title and values of the close and moving average are the framework. A disclaimer or an explanation of the system can be added at the end. The code for the different trade signals will fill out the middle.

Go to the commentary editor. Open the paste functions dialog. Select the “Data” section in the left column. Included on the right will be:

- Security Name
- Expert Name
- Security Symbol
- Evaluation Date

These will be used build the title. Change the alignment to Center. Click the Functions button and select “Expert Name”. Go down to the next line and type: “an analysis of”. On the next line, insert “Security Name” and inside parenthesis insert “Security Symbol”. Go down one more line and

after typing the word “on” insert “Evaluation Date”. The commentary editor should show something similar to this:



Since this is the title, parts should be emphasized. Select the top line and change the font to 18 point. Then select the third line and change it to 14 point. Colors can be add if desired. Bold, Italics, and underlining are also available to draw further attention to some part of the text.

Adding Calculated Values

Press enter to move to the next line and then change the alignment back to Left. Type “Close:” and press enter. On this line, type “40-period moving average:”.

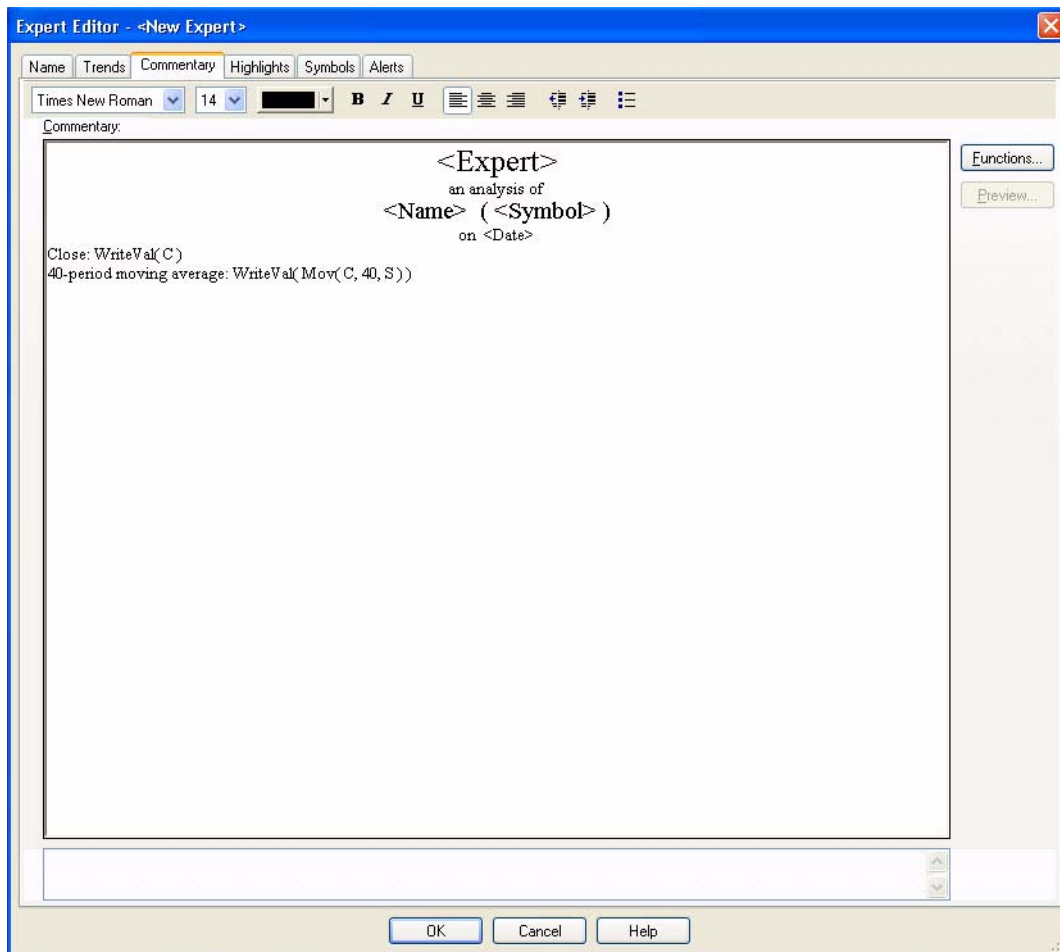
The WriteVal() function is used to display those numbers. WriteVal() can only display numeric values. On the Close line, after the colon, add the text:

```
WriteVal( C )
```

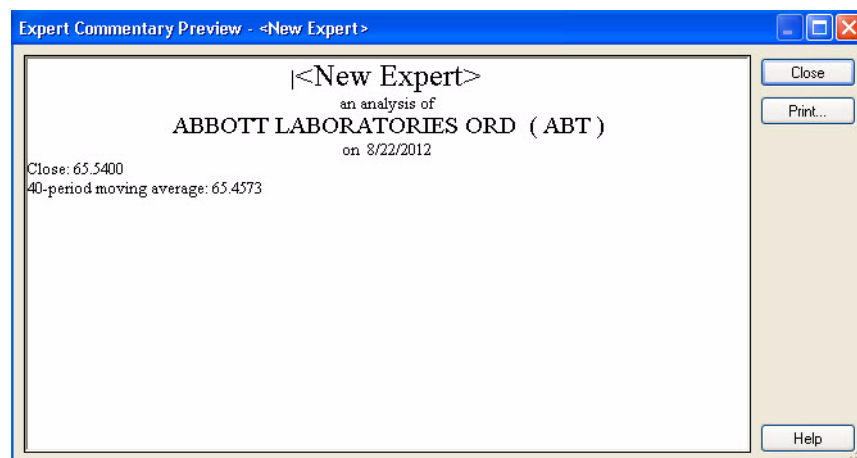
On the moving average line, add the text:

WriteVal(Mov(C, 40, S))

With these changes, the commentary editor should now show something similar to this:



If a chart is open, click the preview button. This will show a window like the one below:



Spacing can be added between the date and closing price line. This would improve appearance but is not required. It would also look nicer to align the numbers for the closing price and moving average. The commentary does not have tabs so this can be done two ways. The first is a rough alignment by visually adding and removing spaces. Most fonts have a different widths for each character so it can be very time consuming to do this and may not be possible to get exactly right. The second method is to switch to a font like Courier that uses the same width for each character. This example will switch to Courier New. Select the text on both the close and moving average lines and then make that change.

The label for the moving average line is rather long so shorten it to “40 period SMA”

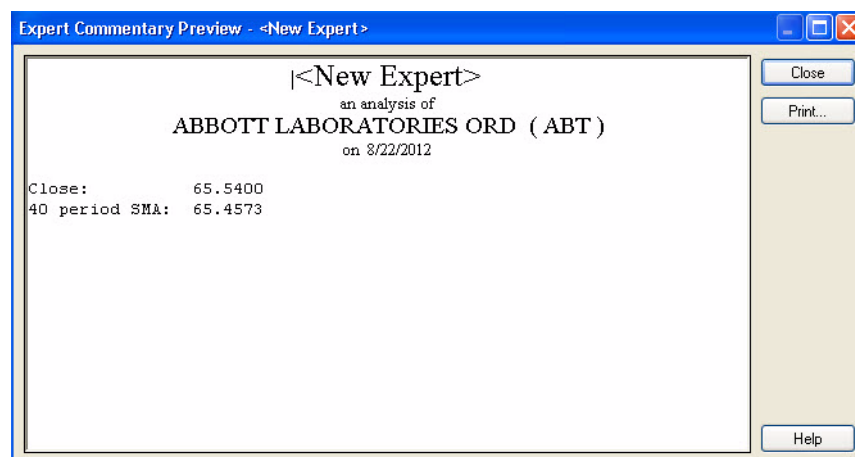
Finally, the WriteVal() function has a second optional parameter. After the formula for the value to be display, add a comma. Now a number can be entered specifying the format for the value. The format is set by the total number of spaces to allow, a period, and then the number of spaces after the decimal period. For example:

```
WriteVal( C, 18.4)
```

That line says to allow 18 spaces for the closing price. 4 of those spaces are after the decimal and 1 is taken by the decimal. The remaining 13 spaces are reserved for numbers to the left of the decimal. Any unused spaces are still inserted. This pads the number format and allows numbers to be lined up even if the size of the number is unknown. The moving average is also padded:

```
WriteVal( Mov( C, 40, S ), 8.4 )
```

Since the label for the moving average is still longer than the closing price, the amount of padding was reduced by the number of extra characters. Now the preview of the commentary looks like this:



Conditional Logic

The logical switch for this system can have any of six different values. The expert should display different information for each possibility. What will be displayed should be planned ahead of time as much as possible. This is like an outline for a document or rough draft. It is not set in stone but is very useful to make sure every thing is covered. The six states are:

- out of the market
- in a long position
- sell signal
- trailing stop
- profit target stop
- time exit

When a exit is found, the expert should list the value of the stop price and the type of exit signal. For the sell signal and the time exit, the price listed will be the close of the bar.

When out of the market, the expert should just state the system is out of the market.

While in a trade, the commentary should list value of the two stops. This commentary will also list how long the trade has been active and how long till the time exit.

The logic for the WriteIf() should also be outlined. This is exactly like a complex If() logic for an indicator except WriteIf() does not require a False result. Since the switch used by the system will only have one value at a time, the system could check for each possible value individually:

```
If trade = 0: out of market
If trade > 0: in a trade
If trade = -1: sell signal
If trade = -2: trailing stop
If trade = -3: profit stop
If trade = -4: time exit
```

To demonstrate nesting of the WriteIf() functions, this expert will use a slightly different logic:

```
If trade = 0: out of market
Else:
    If trade > 0: in a trade
    If trade = -1: sell signal
    If trade = -2: trailing stop
    If trade = -3: profit stop
    If trade = -4: time exit
```

The exact logic used will not matter for this expert, but for some commentaries, it could. It is important to understand that any text inside a WriteIf() is not used in any way unless the conditional text of the WriteIf() exposes it. Consider the following logic:

```
WriteIf( Close > Open, "\
This is a White Candle.
The close of the candle is WriteVal( Close)", "\
This is a Black Candle")
```

The value of the close is only displayed if the close is greater than the open. If that conditional statement is false, the commentary will only display “This is a Black Candle.”

It is of worth to also note the use of the back slashes in the above code. This is a special key that instructs the commentary to not use the next character in the text. When used at the end of a line, the backslash causes the change to a new line to be ignored. This allows spaces to be added to a commentary by the programmer for easier reading of the code without adding unwanted spaces to the final result.

Coding the WriteIf()

With the logic structure defined, the actual formula for the trading system can be added to the WriteIf() functions. For now, the code will use placeholder text. This will be replaced actual commentary text in the next section. Using the nested If() logic defined above, the WriteIf()s would look like this:

```
WriteIf( profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
{ trade exit values
-1 = sell signal
-2 = trailing stop
-3 = profit target stop
-4 = time stop }
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
trade = 0, "\
out of market", "\
WriteIf(profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
{ trade exit values
-1 = sell signal
-2 = trailing stop
```



```

-3 = profit target stop
-4 = time stop }
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
trade > 0, "\
in a trade")\
WriteIf(profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
{ trade exit values
-1 = sell signal
-2 = trailing stop
-3 = profit target stop
-4 = time stop }
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
trade = -1, "\
sell signal")\
WriteIf(profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
{ trade exit values
-1 = sell signal
-2 = trailing stop
-3 = profit target stop
-4 = time stop }
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
trade = -2, "\
trailing stop")\
WriteIf(profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;

```

```

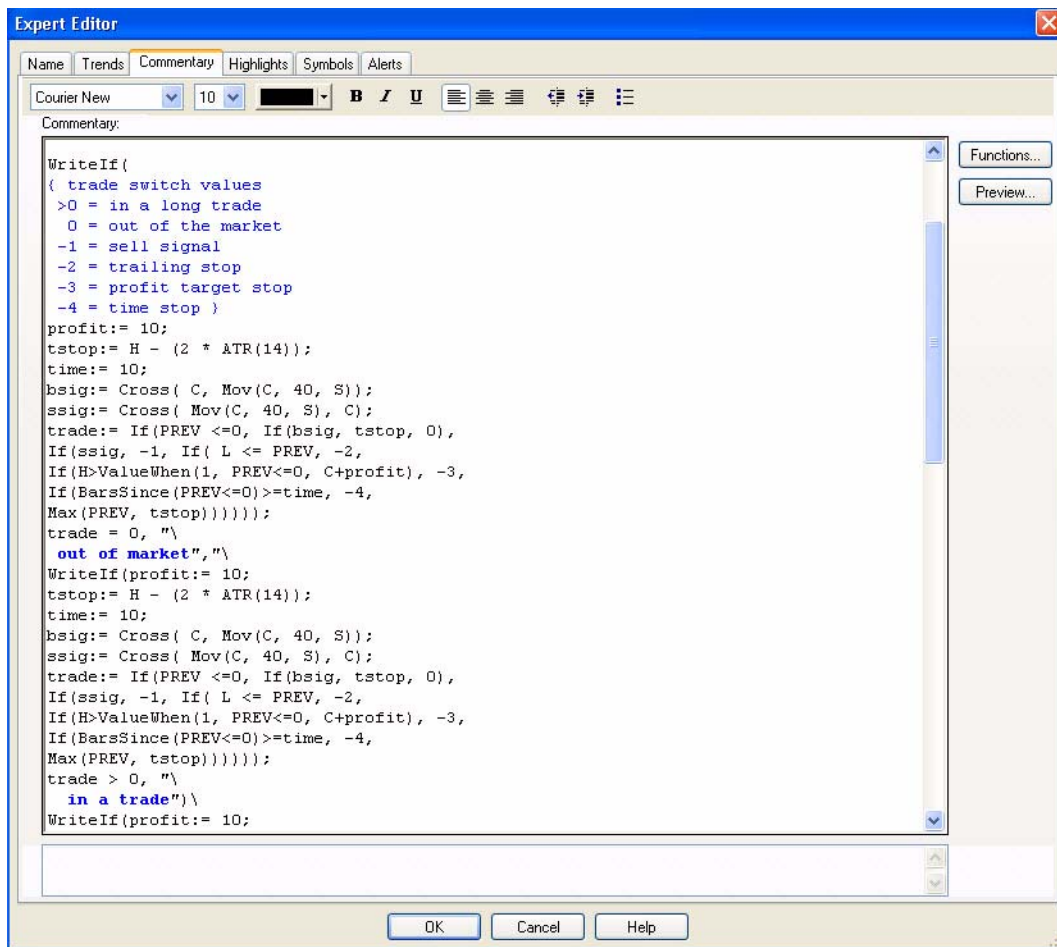
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
{ trade exit values
-1 = sell signal
-2 = trailing stop
-3 = profit target stop
-4 = time stop }
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
trade = -3,"\
profit stop")\
WriteIf(profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
{ trade exit values
-1 = sell signal
-2 = trailing stop
-3 = profit target stop
-4 = time stop }
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
trade = -4,"\
time exit")\
")

```

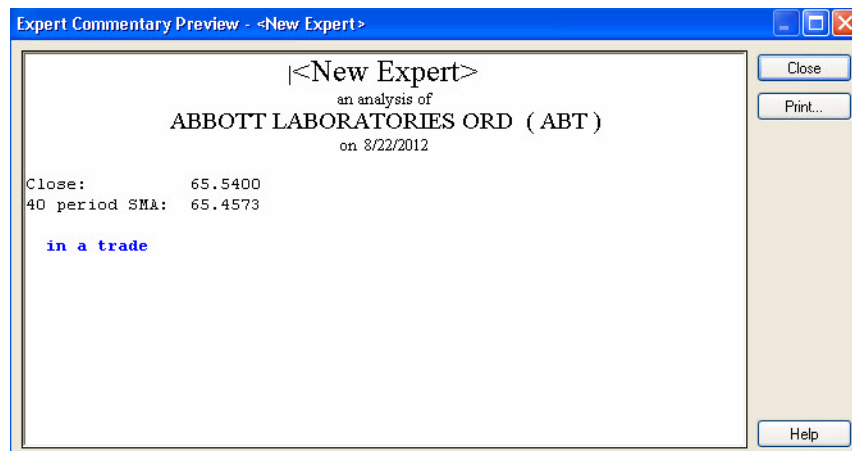
This code is long because it repeats the logical switch of the trading system with each WriteIf(). There are a few things that can be done to shorten this. The most obvious is the comment in the middle of the switch formula. This does not need to be included every time. A single comment at the beginning would be sufficient. The comment could even be made a different color to draw the programmer's attention to it. The users of the expert will never see this unless they edit the expert so the color choice is based on what the programmer feels works best.

A color should also be added to the place holder text to make it easier to spot work still needing to be done. Since red and green text may be used to highlight buy and sell signals, this expert will

use blue text for the comment and placeholder text. The placeholder text was also bolded to make it easier to see. A segment of how the code now looks is show in the screen shot below:



This would be the time to check the commentary against the chart. Look for where the different signals occur. Open the commentary and verify the correct place holder is displayed:



Making the code shorter

A further reduction of the code size is possible but not desirable. If a custom formula was created that returned the value of the systems logical switch, all the code below could be shorted to just Fml() calls:

```
WriteIf(  
{ trade switch values  
  >0 = in a long trade  
  0 = out of the market  
  -1 = sell signal  
  -2 = trailing stop  
  -3 = profit target stop  
  -4 = time stop }  
Fml("switch 1") = 0, "\  
  out of market", "\  
WriteIf(Fml("switch 1") > 0, "\  
  in a trade")\  
WriteIf(Fml("switch 1") = -1, "\  
  sell signal")\  
WriteIf(Fml("switch 1") = -2, "\  
  trailing stop")\  
WriteIf(Fml("switch 1") = -3, "\  
  profit stop")\  
WriteIf(Fml("switch 1") = -4, "\  
  time exit")\  
)
```

There is a trade off for this reduction in code size. Each Fml() call takes longer to calculate than the original formula. If a lot of references like this are used, MetaStock could seem to freeze for a few seconds or longer each time the commentary is updated. There are times this may be needed, but where possible you should put the full code into the commentary.

A second option is to code long logic into a custom DLL. A DLL is not limited by size and can sometimes be faster than the metastock code. Whether the effort to write a DLL is worth the extra time is up to the programmer to decide. Normally, this will not be the case.

Write the Exit States

At this point, the commentary is almost done. All that is needed is the actual text to replace the placeholders put in earlier. The out of market will be expanded to this message:

The system is currently out of the market.
Wait for close to cross the moving average before taking further action

Out of the market should be bolded and raised to a 12 point font size. The O and M will also be capitalized. This draws the readers attention to the status. The rest of the text can be read for further information. This section of the commentary code should now resemble the image below:

```
WriteIf(
{ trade switch values
>0 = in a long trade
0 = out of the market
-1 = sell signal
-2 = trailing stop
-3 = profit target stop
-4 = time stop }
profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
trade = 0, "\
The system is currently Out of the Market.
Wait for close to cross the moving average before taking further action.", "\
```

The “in a trade” state will be skipped for now.

Sell signals are set to exit at the close. The closing price can be listed again but since this is already displayed just a few lines above, this commentary will simply reference it. Change the placeholder text to:

Sell Signal:

The Close has fallen below the moving average
exit the trade at the current Closing price

The first line should be enlarged to at least 12 point. Since it is an exit signal, the color will be set to red:

```
WriteIf(profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
trade = -1, "\
Sell Signal:
The Close has fallen below the moving average
exit the trade at the current Closing price")\
```

The trailing stop should reference the value of the stop. When the logical switch was written, the trailing stop was the value passed to the next bar by PREV. Therefore, Ref() applied to the switch will return the last stop value:

Trailing Stop

The trailing stop was triggered at WriteVal(profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;

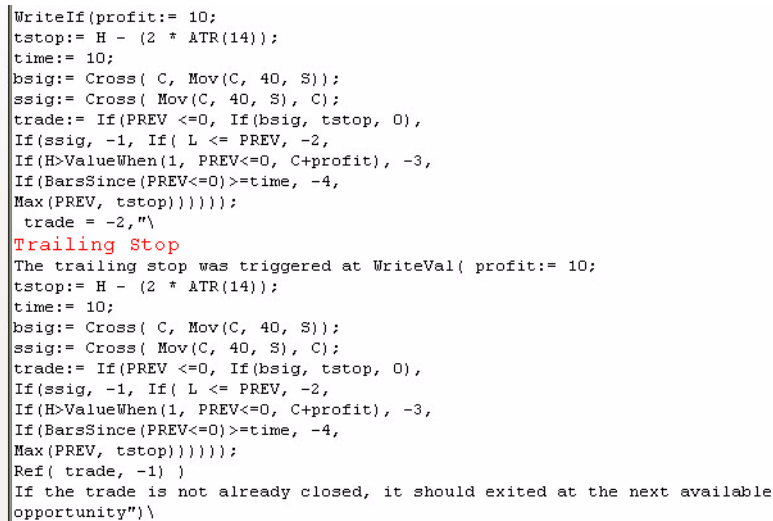
```

bsig:= Cross(C, Mov(C, 40, S));
ssig:= Cross(Mov(C, 40, S), C);
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If(L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
Ref(trade, -1))

```

If the trade is not already closed, it should exited at the next available opportunity

Again, the first line should be increased to 12 point and colored red. The WriteVal() makes the second line longer, but this is still just three lines of text when displayed in the commentary. When added to the editor, it should look similar to the image below:



```

WriteIf(profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
trade = -2,"\
Trailing Stop
The trailing stop was triggered at WriteVal( profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
Ref( trade, -1) )
If the trade is not already closed, it should exited at the next available
opportunity")\

```

The profit target state requires a slightly different logic. The text the commentary will display is:

Profit Target Stop

The profit target of WriteVal() was reached.

If the trade is not already closed, it should exited at the next available opportunity

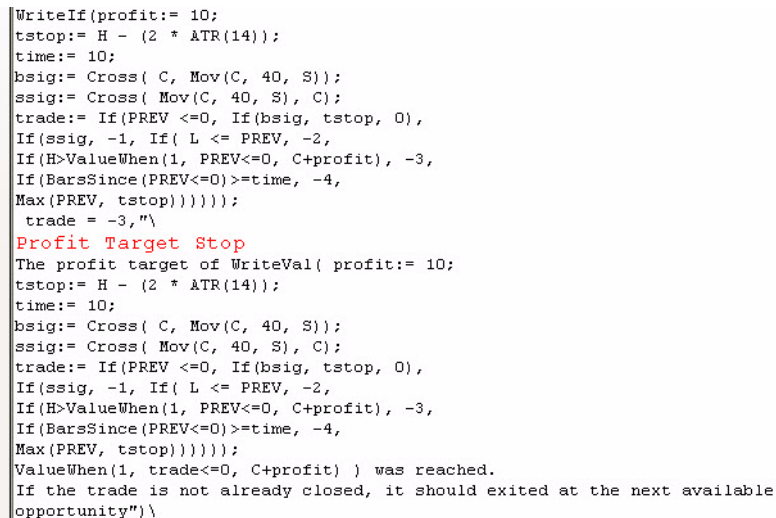
The first line will be increased to a 12 point font and colored red. The WriteVal is the part that will take a little extra work. In the logic of the switch, the value of the profit target was checked by the code:

```
H>ValueWhen(1, PREV<=0, C+profit)
```

PREV refers to the value of the switch so the variable Trade can be substituted: Since the value is needed, the H > is removed from the front of the code changing the text from a condition to a value. Since the current value of trade is less than zero, we need to reference back to the second most recent occurrence. This changes the first parameter of ValueWhen() from 1 to 2:

ValueWhen(2, trade<=0, C+profit)

Putting this into the WriteVal() section, the profit target part of the commentary will look similar to the image below:



```
WriteIf(profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
trade = -3,"\n"
Profit Target Stop
The profit target of WriteVal( profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
ValueWhen(1, trade<=0, C+profit) ) was reached.
If the trade is not already closed, it should exited at the next available
opportunity")\n
```

The time exit does not require any extra values. The place holder is replace with the text below:

Timed Exit

The trade has exceeded the time allocated for it.

If it is not already closed, it should exited at the next available opportunity

The type of exit is again increased to a 12 point font and colored red.

The Last State

The “in a trade” will have several values displayed for it. The basic format of this text will be:

Title Line

Profit Target: WriteVal()

Trailing Stop: WriteVal()

If still open, the trade should be closed after WriteVal() more bars.

The profit target value and the trailing stop values are just duplications of the logic from those exit states. The number of bars left in the trade can be found with the logic:

time - BarsSince(trade<= 0)

The variable time holds the number of bars the trade can be open. On the first bar of a trade, the variable will have the value of 1 since the previous bar was the one that was less than or equal to zero. This count does say how many bars after the current one the trade should remain open.

The title line actually needs one more WriteIf(). On the first bar of a trade, the commentary should state a new trade signal was found. On the following bars, it should say an existing long position is open. The commentary can check if the trade is a new position by looking at the previous value of the switch, just like a buy signal would. This gives a rough logic of the in a trade state as:

```
WriteIf(trade > 0 and Ref(trade <= 0, -1), "\
A new buy signal was triggered today", "\
A long position is currently open")
Profit Target: WriteVal()
Trailing Stop: WriteVal()
If still open, the trade should be closed after WriteVal() more bars.
```

“Buy signal” and “long position” should be raised to 12 point and colored green. The full code for this state, minus the formatting, is:

```
WriteIf( profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
trade > 0 and Ref( trade <= 0, -1), "\
A new buy signal was triggered today", "\
A long position is currently open")
Profit Target: WriteVal( profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
ValueWhen(1, trade<=0, C+profit) )
Trailing Stop: WriteVal( profit:= 10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
```



```

If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop)))));
Ref( trade, -1) )
If stil open, the trade should be closed after WriteVal( profit:=
10;
tstop:= H - (2 * ATR(14));
time:= 10;
bsig:= Cross( C, Mov(C, 40, S));
ssig:= Cross( Mov(C, 40, S), C);
trade:= If(PREV <=0, If(bsig, tstop, 0),
If(ssig, -1, If( L <= PREV, -2,
If(H>ValueWhen(1, PREV<=0, C+profit), -3,
If(BarsSince(PREV<=0)>=time, -4,
Max(PREV, tstop))))));
time - BarsSince( trade<= 0 )) more bars.

```

Final Touches

The commentary is done. It meets all the requirements set forth at the beginning of this chapter. It displays the name of the expert, the security the expert is attached to and the ticker symbol. It also shows the date the commentary is currently being evaluated for. All this is centered at the top.

The commentary shows the values of the Close and the 40 periods simple moving average for the current bar. This is aligned on the left side below the title information

Depending on the current state of the system, the commentary will show messages about being out of the market, in a trade, or which exit signal was triggered. If the system is in a trade, the commentary shows a different message for the start of the trade than for the later bars. While in a trade, the values of the profit target stop and trailing stops are show as is the number of bars remaining till the time exit.

Other text could be added to make the descriptions more elaborate. The layout and style of the commentary can be adjusted and different colors used. None of those require any elaborate coding. The same WriteIf() and WriteVal() functions would be used with whatever conditions and values were desired.

Appendix 1: Simulation Functions

The simulation functions were included in Version 8 for use in the System Tester. They allowed a formula to access current values of an ongoing test. For instance, `Simulation.CurrentBar` would contain the number of the bar the test was currently being looking at. Using this function you could require a signal wait till the 50th bar with the logic:

```
Simulation.CurrentBar >= 50
```

While this function could be replaced by a cumulation of 1, other simulation functions unlocked trading logic previously not possible. However, the simulation functions have several restrictions:

- They only have a value on the current bar. This means you can not use any sort of reference with them including but not limited to `Ref()`, `ValueWhen()`, `Mov()`, `HHV()`, and `LLV()`.
- They only work in the system tester. They cannot be used in any other formula based tool (e.g. the Expert Advisor).
- Their use almost always increases the calculation time of system tests.

The functions are listed below with a short explanation of each.

`Simulation.CurrentPositionAge`:The age of the current position in sell and buy to cover formulas

`Simulation.AccountCash`:The amount of funds in the cash balance of the account

`Simulation.AccountBorrowed`:The amount of funds in the margin and overdraft balances of the account

`Simulation.AccountReserved`:The amount of funds in the reserve balance of the account

`Simulation.LongFundsAvailable`:The amount of leveraged funds available to new long orders

`Simulation.ShortFundsAvailable`:The amount of leveraged funds available to new short orders

`Simulation.CurrentPositionPerformance`:The closing performance of the current position in Sell and Buy to Cover formulas. Closing performance assumes a market order and does incorporate commission costs.

`Simulation.CurrentPositionProfit`:The closing profit of the current position in Sell and Buy to Cover formulas. Closing profit assumes a market order and does incorporate commission costs.

`Simulation.CurrentPositionValue`:The closing value of the current position in Sell and Buy to Cover formulas. Closing value assumes a market order and does incorporate commission costs.

`Simulation.CurrentBar`:The current bar in the simulation

Simulation.PortfolioHighestProfit:The highest summary market value of the portfolio to date. Commissions are incorporated into the calculation.

Simulation.PortfolioLowestProfit:The lowest summary market value of the portfolio to date. Commissions are incorporated into the calculation.

Simulation.LongPositionCount:The number of long positions in the portfolio

Simulation.ProfitableLongPositionCount:The number of profitable long positions in the portfolio

Simulation.ProfitablePositionCount:The number of profitable positions in the portfolio

Simulation.ProfitableShortPositionCount:The number of profitable short positions in the portfolio

Simulation.ShortPositionCount:The number of short positions in the portfolio

Simulation.UnprofitableLongPositionCount:The number of unprofitable long positions in the portfolio

Simulation.UnprofitablePositionCount:The number of unprofitable positions in the portfolio

Simulation.UnprofitableShortPositionCount:the number of unprofitable short positions in the portfolio

Simulation.CurrentPositionPointDifference:The points gained or lost in the current position in Sell and Buy to Cover formulas

Simulation.CurrentPositionSize:The size of the current position in Sell and Buy to Cover formulas

Simulation.PortfolioValue:The summary market value of the portfolio. Commissions are incorporated into the calculation

Simulation.PortfolioProfit:The summary profit or loss of the portfolio. Commissions are incorporated into the calculation.

Simulation.BuyOrderCount:The total number of buy orders, either open or being considered by the trader

Simulation.BuyToCoverOrderCount:The total number of buy to cover orders, either open or being considered by the trader

Simulation.LongOrderCount:The total number of long orders, either open or being considered by the trader

Simulation.PositionCount:The total number of open positions in the simulation portfolio

Simulation.OrderCount:The total number of orders, either open or being considered by the trader

Simulation.ClosingOrderCount:The total number of position closing orders, either open or being considered by the trader.

Simulation.OpeningOrderCount:The total number of position opening orders, either open or being considered by the trader.

Simulation.SellOrderCount:The total number of sell orders, either open or being considered by the trader.

Simulation.SellShortOrderCount:The total number of sell short orders, either open or being considered by the trader

Simulation.ShortOrderCount:The total number of short orders, either open or being considered by the trader

Simulation.CurrentPositionIsLeastProfitable:True if the current position is the least profitable of the current bias in the portfolio

Simulation.CurrentPositionIsMostProfitable:True if the current position is the most profitable of the current bias in the portfolio

Simulation.CurrentPositionIsNewest:True if the current position is the newest position of the current bias in the portfolio

Simulation.CurrentPositionIsOldest:True if the current position is the oldest position of the current bias in the portfolio.

Appendix 2: Glossary

This text tried to use common terms where possible and included definitions of terms that may not have been as well known. The following terms usage in the text may have been outside the reader's previous experience. This glossary is provide to help clarify any unfamiliar usage.

Code: This normally means the instructions or formulas that make up a program. It can also refer to a slang term meaning to write a program or formula in a language a computer or software program can understand. The original term was to write something in computer code. This was later shortened to just "code" something.

Error Trap: Computers program can sometimes result in errors being generated, even if the logic and code are correct. This can happen when unique circumstance arise and disrupt the execution. A common example is a math calculation that results in an attempt to divide by zero. If it is known that such an error may occur, a good programmer will try to prevent it. This is done by adding code to recognize the conditions that would cause the error and adjust the calculations to prevent it. See "Volume with the RSI" on page 13 for an example of how to write a divide by zero error trap.

Flag: When writing programs, the term flag refers to the action of marking an event. This is similar to a football referee throwing a flag on the field to mark where a foul happened. In programming a flag is a switch that is either on or off. When it is on, it designates some event or condition is present. The flag is then reset (or turned off) when the condition no longer exists.

When writing computer programs, a boolean variable that can only be True or False, is normally used for flags. In MetaStock's formula language, a flag must be done with a 2 state logical switch.

Logical Switch: A logical switch is very similar to a mechanical switch. The simplest version of this is a light switch. It has two states, either on or off. However, there are more advanced versions, like the gear shift for the transmission of a car or truck. These "switches" an have multiple settings.

In MetaStock's formula language, a logical switch is constructed by nesting If() functions. See "The Logical Switch" on page 38 for a more detailed explanation of creating these.

Nesting: In programming, to nest something is to put it inside something else. This similar to, and possibly taken from, the Russian nesting dolls. An example of a nested function is a rate of change calculation applied to a moving average function:

```
Roc ( Mov ( C , 20 , S ) , 5 , $ )
```

In this case, the Mov() function is nested inside the Roc() function.

Optimization: Optimization is a method MetaStock uses to help create a better trading system. The MetaStock system tester can run the same test multiple times. Each time, it will change the value of a numeric variable. When all possible combinations of variable settings have been tested, it reports the ones that were most profitable. There is a danger in over optimizing a system. For more information on how to optimize and its pitfalls, please consult the MetaStock Help.

Parsing: Many times a computer will be given a long series of letters and/or numbers. For example: 51904. This number could be zip code, a customer number, the price of a stock or index. In and of itself, the number is meaningless. However, if the proper instructions are provided in the formula or computer code, the series of numbers can be parsed (or separated) into individual values for use in the code. To continue with the example number, if we started from the right and sectioned it off into 2 digit pairs, we would get the following values: 05 / 19 / 04. This could then be used as a date. Other results are also possible, depending on how you parse the values.

The need to parse a series of numbers arises when the methods of inputting value becomes a constraint. For example, MetaStock allows only six Input() functions per indicator. If you needed to supply 7 values, you could combine two and then have the formula separate them back into individual values. This requires the numbers be entered in a specific manner. If the input is not formatted correctly the parsing process will not produce the expected values and the formula will either fail or plot something different than desired.

Simulation Functions: MetaStock has some formula functions that only work in the MetaStock Enhanced System Tester (MetaStock version 8 or later). These functions are designed to report values about the status of a system test while it is running. Some example of the functions outputs are how many bars a trade has been open; the number of trades currently open, and the amount of equity available to open another trade. For complete details on these functions and what they do, see “Simulation Functions” on page 109