EASING INTO EASYLANGUAGE

The Foundation Edition

George Pruitt

ISBN-13: 9781234567890
ISBN-10: 1477123456

Cover design by: Art Painter
Library of Congress Control Number: 2018675309

Printed in the United States of America

To All the Dreamers Who Want To Validate Their Grand Schemes

During the time I was writing this book, my friend and great market technician / programmer / software developer ***Murray Ruggiero*** passed away. I dedicate this book to him as well!

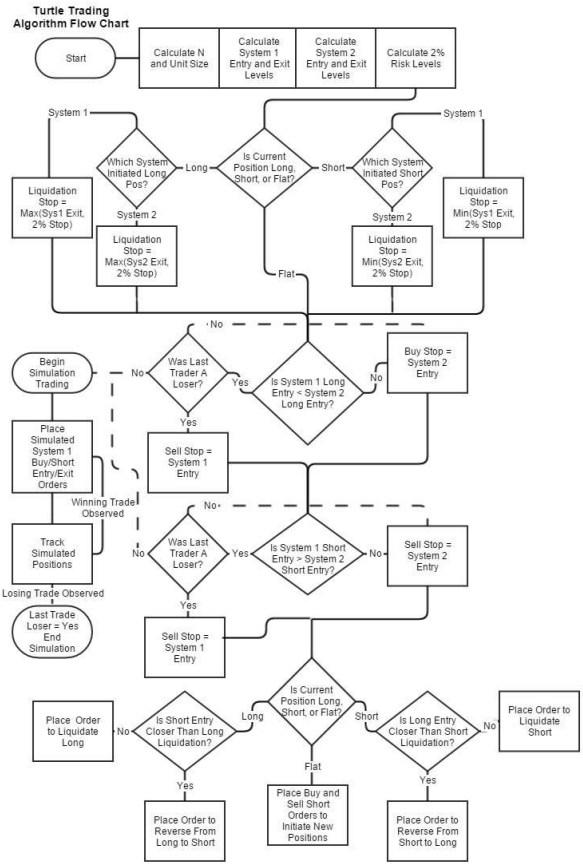
Table Of Contents

EasyLanguage – What Is It?

An algorithm must be seen to be believed.
Donald Knuth

EASYLANGUAGE is an immensely powerful and versatile language, and you should be able to program a vast majority of your trading ideas with it. However, most new users of the language have described it as anything but "EASY." In this book, we will be utilizing what is called the "classic" version of the language. Discussions will be centered on the procedural paradigm of the language. In addition, most of my focus will be on strategies and how to program this type of analysis technique. I will provide a few snippets of code on Indicators, ShowMes and Paintbars. Object programming is well beyond the scope of this book, but do not worry, the lan-

guage is still quite powerful in its original iteration. The creators of EasyLanguage wanted an easy-to-use medium to program ideas and tied prefabbed functions and indicators together so one could get up and running rather quickly. In my opinion, they have succeeded in doing so. If you have any experience with programming languages you might notice a close similarity between EasyLanguage and PASCAL. I think the first EasyLanguage compiler was some old derivative of a PASCAL compiler from the '70s or '80s. PASCAL was created to teach data structures and good programming style. With this in mind it is definitely a great language for newbies. Eventually, with practice you will be able to program an algorithm similar to



the image on the left. The code for this algorithm is actually downloadable from my website.

EasyLanguage falls under the **sequential one market one day at a time paradigm**. TradeStation, without Portfolio Maestro, tests by applying a strategy to individual charts and then examining the results from each chart. Portfolio Maestro is a standalone program that you can acquire from TradeStation that will combine the results of your algorithm after it is applied to a portfolio of markets. This software will give you the impact of your algorithm when it is applied and combined to more than just one single market. However, you still need to get your algorithm into a form that either TradeStation or Portfolio Maestro can interpret.

Assumptions

This book assumes you are familiar with creating charts in TradeStation and inserting a strategy or indicator into the chart. Also you should know how to launch the EasyLanguage editor, either through TradeStation or the TradeStation group in the Start menu.

Foundation Edition Only Deals With Daily Analysis

This Is A Beginner's Book or a Refresher

Only daily bar strategies and indicators will be reviewed in this book as its a beginners tutorial. This book will provide a great foundation to dig deeper into more sophisticated analysis and intraday strategies later on. If you have ever stared at an empty EasyLanguage screen and wondered how and where to start this book is for you.

Easylanguage Editor

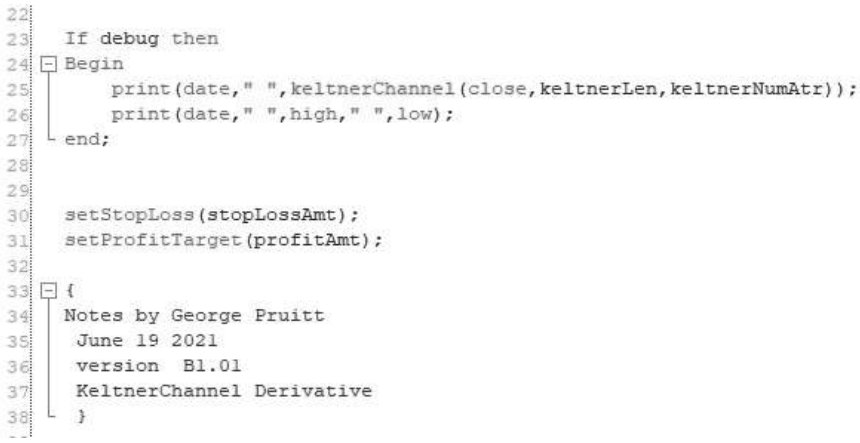
Also known as the TradeStation Development Environment or TDE is where all the magic happens. All your code will be typed into this editor (word processor-like), verified (compiled), and stored. Most programming languages allow you to use your own favorite code editor (I love Sublime), but not EasyLanguage. You could type your code into an external editor and then paste into a blank TDE analysis technique, but that requires additional steps. The additional steps might be worth it though, because you would have a text based version of your code that you could back up. The TDE stores everything in one big library glob and if you lose that glob, then you lose

everything. However, TradeStation and the TDE allow multiple ways to back up your work.

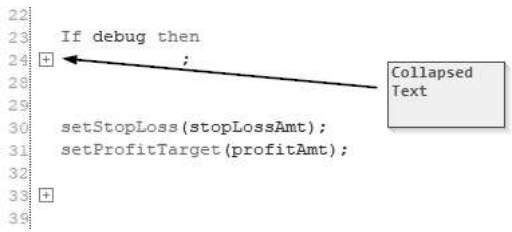
The TDE editor is a very powerful editor that provides some very high end options. You can access the TDE independent of TradeStation by going to the TadeStation group on the Start menu or you can launch it directly from TradeStation's Trading App pane. The following details some of those high end options.

Code collapsing (TDE Outlining)

This feature allows you to hide code that might be distracting you in the editor window. The following shows the text before and after collapsing



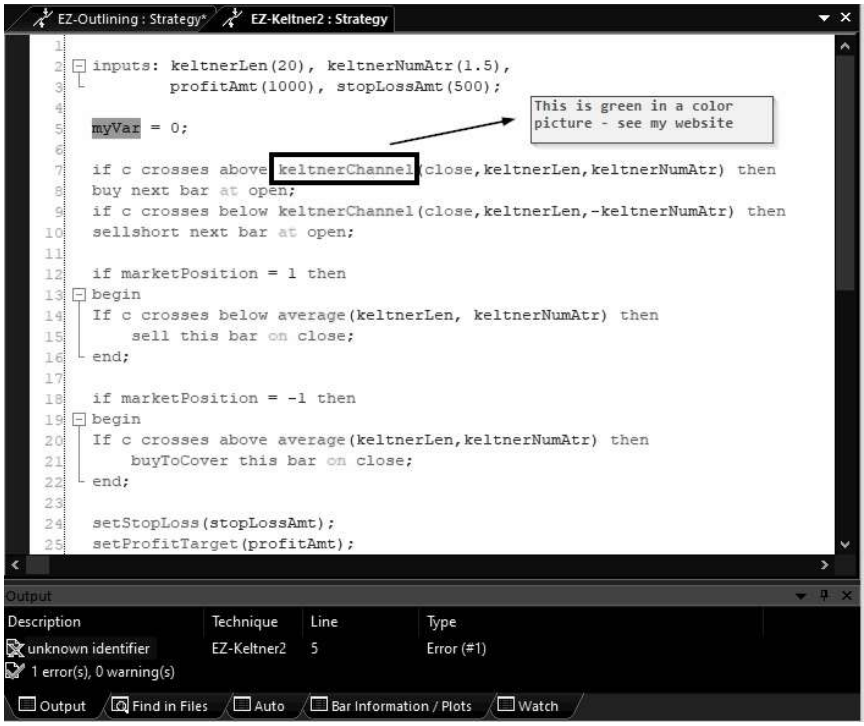
Now with code collapsed



This is a nice feature, especially for very large coding projects. If you need to learn more about outlining go to the **Outlining** menu item in the **Edit** menu.

Line numbering and Syntax Coloring

Having line numbers is great whenever you have a syntax error, because the TDE will tell where the error is located. Also Syntax Coloring helps you by coloring the text it recognizes. Take a look at the different colors of the syntax from my website (www.georgepruitt.com) and the error message from my intentional syntax mistake.



Notice how the Output window directs you to line number of the syntax error and its type. I didn't put the user defined myVar into a **Vars**: section. In full color, the KeltnerChannel function is green and the rest of the reserved words are an orange like color. If I type KeltnerChan(blah, blah, blah) it will not turn green, because it doesn't match the actual function name. This lets you know that you have it wrong. If you have installed TradeStation and opened the TDE your color scheme will be different - basically it will by black text on a white background. You will soon discover if you spend time in front

of this color scheme your eyes will start doing crazy things. The dark background with lighter colored text is a favorite of many programmers including me. You can change your scheme by going to the **Options** menu item on the **Tools** menu in the TDE. Make sure you have **Syntax Coloring** checked and then start playing around with the colors.

Auto Completion

This is a nice feature when are just starting out. If you type a function name and it turns the appropriate color a function tip dialog will open and let you know what the function requires to return the desired value. Here the **BollingerBand** function needs Price, Length and the NumDevs (number of deviations).

Modified Camel Case Style

You will notice that my variable names and function calls have a mix of upper and lower case letters. My style is called a modified camel case syntax. Here are some examples.

myEntryPrice - first letter is lower case and each subsequent word in the variable name is upper case.

longDateOfEntry - notice the upper and lower cases?

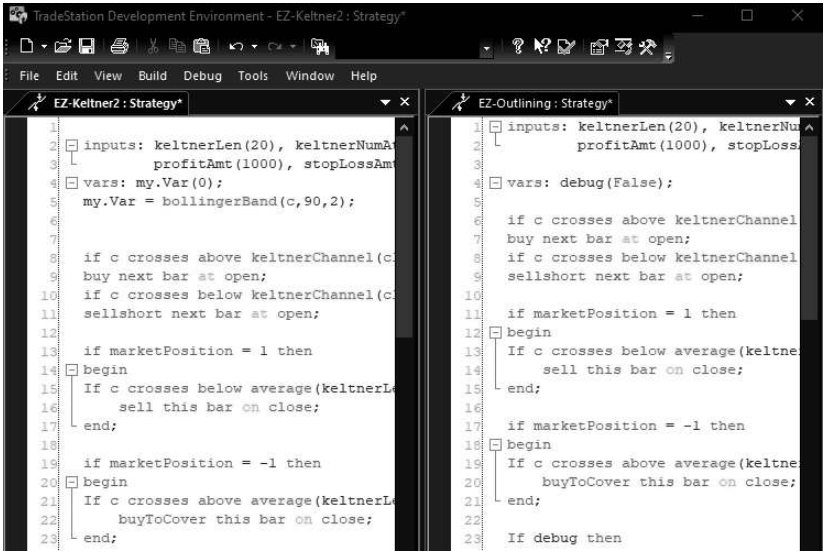
This is just my way of naming variables. Some people like:

- **my_entry_price** – use underscores – this is okay – old school
- **my.entry.price** - use periods - don't do this
- **myentryprice** - all lower case - okay
- **MYENTRYPRICE** - all upper case - okay but stop shouting
- Create your own style. You can't use anything like mathematical notation like -, + , /, * in your variable names. And you variable names cannot start with a number.
- **1stEntryLevel** - nope
- **EntryLevel1** - okay because numbers are fine after the first letter and you can capitalize the first letter of a variable name - yuck! Why yuck? I sometimes use upper case as the first letter for predefined names or function calls, but most of the time I use the modified camel back.

Spend Time Familiarizing Yourself With The TDE

There are a ton of things you can do with the editor - one of my favorites is putting all the files I am working with into tabs-like Excel worksheets. I like to also split the screen when working on two similar analysis techniques.

You do this by grabbing the name of the analysis technique from the tabs and dragging it into the body of the currently open file. The TDE will ask if you want a new **Horizontal or Vertical Tab Group**; I prefer vertical split screens.



Ever so often make sure you export your latest analysis techniques through the TDE **import/export** option on the **File** menu. This will create an **ELD** (EasyLanguage Archive) and remember to put it somewhere in the cloud or on a flash drive. Copying the actual code into NotePad++ or Sublime and archiving a text version is a great idea too.

Back Testing Theories And Future Leak

There are basically three back testing theories available through many commercially available testing platforms.

The first is what I all the **Vertical Sequential Multiple Market Spanning** (VSMMS) method –

Market by Market Proces					Portfolio
Date	Crude	Corn	Euro C.	Bonds	Results
1/4/2000	79.48	553.5	1.06335	80.9688	NO
1/5/2000	78.84	553.5	1.06505	80.0313	NO
1/6/2000	78.71	554.25	1.06305	80.75	NO
1/7/2000	78.15	557.5	1.06135	81.2813	NO
1/10/2000	78.6	559	1.05855	80.9063	NO
1/11/2000	79.7	557.75	1.06545	79.9375	NO
1/12/2000	80.21	566	1.06405	79.75	NO
1/13/2000	80.62	571.25	1.05815	80.625	NO
1/14/2000	81.95	569.5	1.04665	80.0938	NO
1/18/2000	82.78	570.5	1.04445	79.5625	NO
1/19/2000	82.86	571.25	1.04455	79.875	NO

here you go through one market's history completely and then move onto the next market. Look at the data in the table, where the dates are rows and markets/stocks are columns. The testing engine starts looping at Row 1 and Column 1 and then moves down to Row 2 all the time keeping track if your logic based criteria has been met and if so keeping track of the market's position, open trade equity, closed trade equity, entry price, number of days in a trade and many other things. Once the number of rows of Column 1 have been exhausted all the metrics for the market in Column 1 are calculated and stored for later use. The testing engine then moves to Column 2 and

does the same thing until all the rows are exhausted. Once all the Columns or Portfolio have been exhausted, the portfolio analysis logic then starts to accumulate all of the stored values for each column. This is how Portfolio Maestro sort of works, but once the trades are established, then you can play all sorts of games with position sizing and equity curve trading. The second theory **Vertical Sequential Single Market Spanning** (VSSMS) is how TradeStation works. Basically, it starts at Row 1 and Column 1 and stops as soon as all the rows are exhausted. Plot a chart and apply an algorithm to it and see what happens. You can then create a workspace of several markets and apply the same algorithm to each and click to **View the Strategy Performance** report. It does not matter how many markets are in your workspace, the Performance Report will only show the metrics for the active chart. This is an important distinction, VSSMS versus VSMMS. If you are simply interested in testing one market then analyzing one market at a time is sufficient. However, if you want to create a portfolio and apply an algorithm to all the markets to see if the algorithm is robust then you must use the multiple market spanning technique; create an algorithm and then apply it to a basket of markets and see what the performance is across the entire portfolio. In this method, the back tester simply applies the algorithm to one market at a time and

then accumulates the results and passes this off as a portfolio analysis.

In most cases this form of portfolio testing is fine and it does provide insight into how robust your trading algorithm is by applying it to data it has not seen. The third theory, one I have coined as **Horizontal Sequential Multi-Market Spanning** (**H**SMMS), processes data across the portfolio on day at a time.

Day by Day Process					Portfolio
Date	Crude	Corn	Euro C.	Bonds	Results
1/4/2000	79.48	553.5	1.06335	80.9688	YES
1/5/2000	78.84	553.5	1.06505	80.0313	YES
1/6/2000	78.71	554.25	1.06305	80.75	YES
1/7/2000	78.15	557.5	1.06135	81.2813	YES
1/10/2000	78.6	559	1.05855	80.9063	YES
1/11/2000	79.7	557.75	1.06545	79.9375	YES

This method allows for dynamic portfolio allocation at the end of the trading day. In other words, you know all of the positions, equity, risk metrics and anything else you might want to program on every historical bar in the entire portfolio. Some applications such as AmiBroker, TradingBlox, PortfolioMaestro, TradersStudio and my

own Trading Simula-18 incorporate some form of this type of back testing in their platforms.

Future Leak

If you can look at tomorrow's price action and make a trading decision today, then you have access to Future Leak. In terms of back testing this is an immensely powerful tool, but if misused on purpose or by accident it is extremely dangerous. I have programmed thousands of systems and several programmers took advantage of the leak and produced amazing equity curves; those that slope up at a 45-degree angle with truly little deviation. I could see the misuse of Future Leak a mile away. From an ease of programming perspective, being able to look into the future to see exactly which signal would be hit and then make a decision based on that information is quite helpful. However, its use requires an in-depth knowledge of programming. If new programmers and new system developers have this tool at their beck and call, then they might create trading systems that have little chance of success in the future. Here is the most simplistic abuse of Future Leak:

```
If High > Open then Buy Open
```

Here is one that might even escape a seasoned programmer If High > BuyLevel then Buy at BuyLevel. What if the High equaled (=) the BuyLevel, then the trade

would not take place. In other words, you would never buy the high of the day and if you have traded for a period of time you know this happens frequently.

The creators of EasyLanguage eventually eliminated the concept of Future Leak in the software. Since all data is historical it is quite easy to inadvertently look into the so-called future and use that data to create a trading algorithm. You will create an incredible equity curve, but you will be cheating, and it will fall apart. In an attempt to prevent Future Leak EasyLanguage only allows you to look at the opening price and date/time of the next bar that is being analyzed and of course all prior data. You can't program the idea of if today's close is greater than today's open then buy today's open. Imagine you are an exceedingly small creature, and you can crawl around the chart on the screen. EasyLanguage will only let you crawl to the left of the screen or back in time from you current location. When testing, the language always sets you on the close tick of the of the bar that just closed. This is where you must operate and make your trading decisions for the upcoming day or bar. You can sneak a peek at the next day's bar, but only its open, date and time. This an especially important concept to understand, because all of your orders will be placed either for the current bar's close or sometime/some place on the next bar. In the following tutorials this concept will become exceedingly

evident. But before we jump into some coding, let's take a look at the best way to use the Kindle or eBook version of this book alongside TradeStation or Multicharts.

Naming Conventions Used In This Book

You will soon discover that there is a ton of computer code (EasyLanguage) in this book. And all of this code will be explained in the sections where the code is embedded. The code will be in a separate font – I used a monospaced font so it would be easy to recognize. Monospaced means each character or letter/number covers the exact same horizontal spacing so things line up nice and pretty. Here is an example of a monospaced font:

```
Hello I am George Pruitt and I love to code in EasyLan-
guage.
If C > C[1] and
      C > C[2] and
      H < H[6] and
```

You shouldn't have any problem recognizing what is regular text and computer code. Also I will try and use **BOLD when introducing an EasyLanguage keyword, built-in function and other significant words .** I will probably only do this the first time I introduce the word, as it makes reading more difficult if most of the words are bolded. In EasyLanguage you can use abbreviations for some keywords such as C for **Close.** I will not bold the abbreviations in the explanation of the code. If you ever

have any doubt if a word is a keyword, you can simply search for the keyword in the TDE Help under **EasyLanguage Reserved Words and Functions.** Or you can email me too! **george.p.pruitt@gmail.com**

BEST WAY TO USE THIS BOOK ALONGSIDE TRADESTATION or MULTICHARTS

The Kindle Create software formats all the text in a book to work with several different Kindles or eBook readers. I like to use a Kindle Reader on my computer and then follow along with whatever software the Kindle is describing and the actual software. This is the best practice with this book. This book was uploaded to Amazon as a PDF so that the format would be maintained. Because of this, some of the features that you may be accustomed to in your ePub reader may not function. You may (I am hoping) or may not be able to copy the EasyLanguage code from the Kindle reader directly into your EasyLanguage editor - it may or may not work. All the code files are located on my website.

www.georgepruitt.com

in one big ELD
also any workspaces will be there as well

And more importantly all the videos associated with several of the tutorials will be there too. I recorded over four hours of video to complement each tutorial in the book.

So I would import the ELD into my EasyLanguage editor and then start reading this book - opening code files as you come across them. I will make sure I label the file names appropriately so you will have no problems synching up with this Kindle book. If you are working with MultiCharts you will not be able to open the included workspaces - just build them from scratch. As mentioned, many of the Kindle features will not work with this type of book, because of the formatting this is used to show the EasyLanguage code. Kindle is a great outlet for fiction, but not so much for textbook like layout. I will pepper videos throughout the book so you will need to click on the links or copy the link addresses into your browser of choice. I tried to place the links in appropriate locations in the tutorials, but you may want to skim ahead to the link if you are having problems understanding the current concept. If you are reading the paperback version, the actual URL will be provided.

TUTORIAL 1: TURTLE STYLE TRADING ALGORITHM (BARE BONES)

The Turtle trading algorithm is an amazingly simple concept and can be considered the "Hello World" program in the Quant (trading programming) community. Let us work on the core of this algorithm first, and then we can add more bells and whistles as we master the concepts along the way.

EZDonchian1 – Your First Official Strategy

Create a new Strategy within your EasyLanguage Development Library and name it EZDonchian1.

Type the following code into your empty editor window:

```
//EZ_Donchian1
Buy("DonchBuy") next bar at Highest(High,40) stop;
SellShort("DonchSell") next bar at Lowest(Low,40) stop;
```

This code places stop orders for the next bar based on the highest/ lowest highs/lows from the prior forty days. This is acceptable code because we are placing an order without peeking at the next day's price action. EasyLanguage relies heavily on its vast library of functions and indicators. Here we are using two functions: **Highest** and **Lowest**. A function is simply a sub-program that returns a value based on the information that is passed to it. Don't worry we will discuss functions in a later tutorial. Here we are asking the functions to return the highest high of the past 40 days and the lowest low of the past 40 days. These functions require two inputs: 1) the prices we want the values culled from and 2) how many days to look back. We then tell the computer to buy/sell short if the next bar penetrates these levels on a stop order.

The link to Tutorial 1 video is:

https://vimeo.com/578505056/9cb1cfafb7

This is a complete trading strategy and at one time it was a good algorithm. After you type in the code go under the **Build** menu and click on **Verify**. The Verify process first saves your code and then compiles it. If there are any syntax errors then you will be provided with a list of them. If you receive any errors, just go back to your code and double check for typos. Notice the name enclosed in parentheses and quotes that follow the order directives

Buy and **SellShort**. These strings or words inform TradeStation to label the trades generated by these directives with these names. It is always best to name your entries and exits (we will discuss shortly) for future reference. Naming your entries and exits help keep things organized, with debugging and as you will see in Tutorial 7 it provides the link to customize which exit goes with which entry. Did you also notice the semicolon **(;)** that ends each line of code? This is a line termination character that informs the complier (another name for the Verifier) that a line of code has been completed and to look for another line of code to process. Always end your lines with a semicolon (;). The keywords to initiate long and short entries are:

Buy - enters a long position
SellShort - enters a short position

And the keywords to exit longs and shorts are:
Sell - exits a long position
BuyToCover - exits a short position

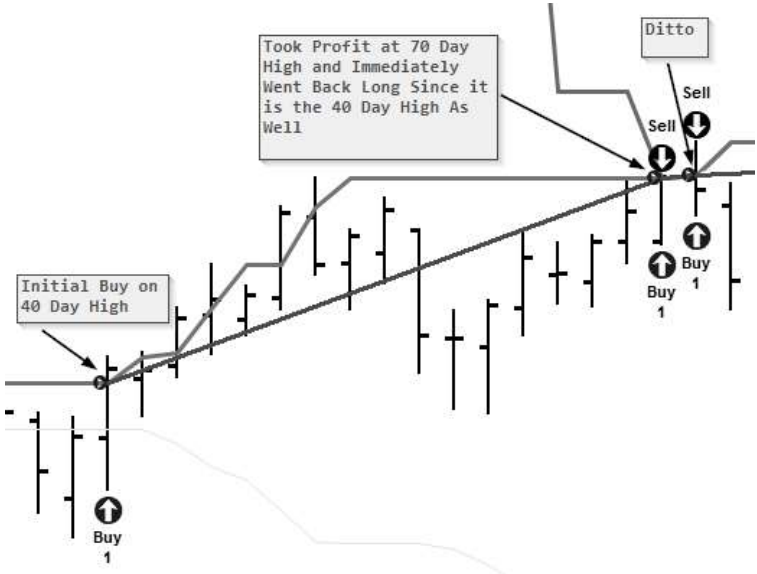
It is important to understand that unless otherwise stipulated, all order directives are placed simultaneously. Unless you branch around an order directive, using logic to prevent the order directive from being executed, all or-

der directives flow into TradeStation's order placement module simultaneously. Just because you put the Buy order directive first doesn't mean it has precedence. TradeStation acts as the broker you call over the phone. If you don't provide additional information, Buys and SellShorts and Sells and BuyToCovers will flow into the market no matter the order they were phoned in. This is important to understand and why in many cases you would want to precede your order directives with some type of additional logic. We will discuss conditional branching in a later chapter, but the concept is important enough to mention now.

```
Buy next bar at highest(h,40) on stop;
Sell next bar at highest(h,70) on limit;
```

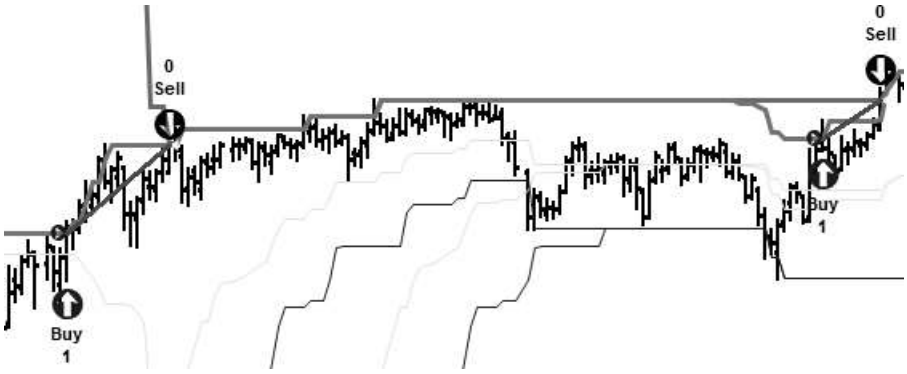
Assuming the market just closed, what do you think would happen if you told your broker to Buy tomorrow on a stop at a 40-day high (a price you would already know) and take profits at a 70-day high on a limit and the market actually moves to the 40-day high level. Your broker would call you at the end of the day (assuming you had taken a time machine back to a time when trading pits still existed) and tell you that you were a buyer at the 40-day high. You hang up happy because you get your price. Remember you are placing these two orders everyday no matter what. Fortune becomesyour friend and on the

next day the market moves way up to the 70-day high level. You see this and await a congratulatory call from your broker. Boy aren't you smart. Well the broker calls you at the end of the day and says you SOLD 1 at the 70 day high for a profit (yoohoo!) and then BOT another 1 at the 70 day high. You gasp, "Huh?" So you thought you were going home flat with a profit, but no you are going home Long 1 with a profit. You begin to argue with the Broker [his name is Hal btw] and say, "I didn't want you to do that!" But HAL calmly responds back and says "I am Sorry Dave – but I can do that. You told me exactly what to do." You fire back, "You know I just wanted to take a profit!" Hal replies back, "I am Sorry Dave, but I followed your directions. I am placing the same order for tomorrow too! Dave, this conversation can serve no purpose anymore. Goodbye"



Remember you are placing orders for tomorrow without any information from today – if you follow the above code that is. Shouldn't the computer know your intention here? No, with TradeStation, without further directions, you will not get a chance to process this information and you will place the exact same order every day. What will happen on the next day? More than likely the market will gap above the 40 day high and the 70 day since it is already there and you will exit and reenter long at the open simultaneously again. This is not what you want, but it is what the computer understood as your intentions. You might have wanted the market to pull back first and get in at a better price. If so, then you have to specifically SPELL THIS OUT to the computer.

This type of logic error is the one I run up against all the time. I will spill the beans here on how to fix this. You may not understand this right now but you will.




```
value1 = highest(high,40);
value2 = highest(high,70);
if c < value1 and
value1 < value2 and
marketPosition <> 1 then
      Buy next bar at highest(h,40) on stop;
Sell next bar at highest(h,70) on limit;
```

This time if the current day's close is below the 40-day high, you call Hal, your broker, and tell him, "If I am not already long and the 40-day high does not equal the 70 day high, then buy tomorrow at the 40-day high on a stop. If I get long during the day or are already long, then sell (cover long position) at the 70-day high on a limit, Thank you." You hangup and say to yourself, "Hal you are a dumb A$$!"

The EZ_Donchian1 code uses a totally different entry mechanism so we don't need to worry about Hal. Go ahead a create a continuous chart of @CL (TradeStation Symbology – yours might be different) or your favorite stock and insert the EZ- Donchian1 strategy. Make sure you specify 10 years of history and use the "@" prior to the symbol name, if you are using a futures contract. This informs TradeStation to create a continuous contract of crude oil data. Once the chart is created with the requested data simply right click the chart and select **Insert Strategy**...

EZDonchian1 Results 10 Years Of @Cl Crude Oil

TradeStation Performance Summary

	All Trades	Long Trades	Short Trades
Total Net Profit	$72,160.00	($4,350.00)	$76,510.00
Gross Profit	$154,840.00	$38,130.00	$116,710.00
Gross Loss	($82,680.00)	($42,480.00)	($40,200.00)
Profit Factor	1.87	0.9	2.9
Roll Over Credit	$0.00	$0.00	$0.00
Open Position P/L	$22,470.00	$22,470.00	$0.00
Select Total Net Profit	$72,160.00	($4,350.00)	$76,510.00
Select Gross Profit	$154,840.00	$38,130.00	$116,710.00
Select Gross Loss	($82,680.00)	($42,480.00)	($40,200.00)
Select Profit Factor	1.87	0.9	2.9
Adjusted Total Net Prof	$12,692.62	($32,921.79)	$21,846.78
Adjusted Gross Profit	$114,860.48	$23,718.21	$75,446.78
Adjusted Gross Loss	($102,167.86)	($56,640.00)	($53,600.00)
Adjusted Profit Factor	1.12	0.42	1.41
Total Number of Trades	33	16	17
Percent Profitable	45.45%	43.75%	47.06%
Winning Trades	15	7	8
Losing Trades	18	9	9
Even Trades	0	0	0
Avg. Trade Net Profit	$2,186.67	($271.87)	$4,500.59
Avg. Winning Trade	$10,322.67	$5,447.14	$14,588.75

TUTORIAL 2: USING INPUTS TO CHANGE FUNCTION PARAMETERS

Testing this algorithm with the 40 days hard coded in the function calls will get the job done. However, if you want to change the parameter you will need to go into the code and make the changes then re-Verify. This would be horribly redundant. Always remember computers were created to reduce redundancy. Creating inputs in Easy-Language allows the code to interface with the user and prevents any modifications to the software. This interface is also necessary if you provide the source code in a black box format and want to allow a user to change system specific parameters.

Interfacing Algorithm with Inputs

Creating inputs and interfacing with your analysis technique is easy and accomplished by using the **Input** or **Inputs** statement. It is not necessary to put the inputs at the top of the code listing, but it is good programming style. We will not go deep into programming style just

right now, but it is something that should be considered. Here is a quick overview – the two most important aspects of style are consistency and modularity. Keeping inputs at the top of the code on every algorithm you program provides consistency and modularity at the same time. Modularity is a component of style that helps others reading your code (and yourself in many cases) understand what they (or yourself) are actually looking at.

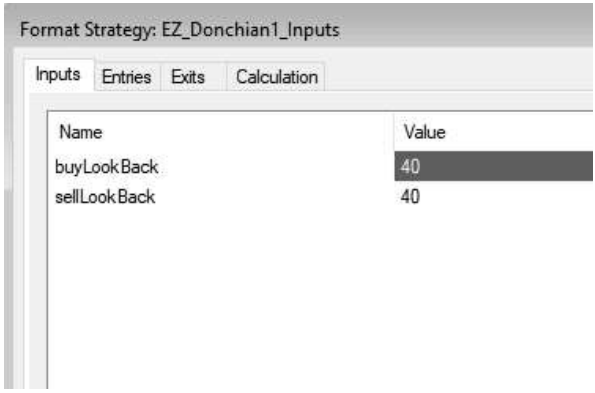
Here is the code listing incorporating inputs into our bare bones Donchian algorithm (notice the colon (:) after the keyword inputs):

```
//EZ_Donchian1_Inputs
inputs: buyLookBack(40),sellLookBack(40);

Buy("DonchBuy") next bar at Highest(High, buyLookBack)
stop;
SellShort("DonchSell") next bar at
      Lowest(Low,sellLookBack) stop;
```

See how I split the SellShort order directive across two lines. I did this for readability purposes only. However, this is perfectly fine since EasyLanguage knows where the line actually ends.

Here is the link to the video for Tutorial 2:

https://vimeo.com/578168730/74d646c772



Change your existing code to reflect the current version. Verify it to make sure you have typed it in correctly. Now if you want to change the inputs all you have to do is Format the strategy by right clicking the chart where the strategy is applied and selecting **Format Strategies:**

What Have We Learned From Tutorials 1-2 ?

- How TradeStation does not peek into the future even when back testing.
- How to use EasyLanguage to place a stop order for the next bar based on the return value of a function.
- How to call a function.
- It is best practice to name each entry and exit signal directive.
- How to utilize Inputs as parameters for functions and change them from the **Format Dialog**.

TUTORIAL 3: SIMPLE BOLLINGER BAND UTILIZING INPUTS

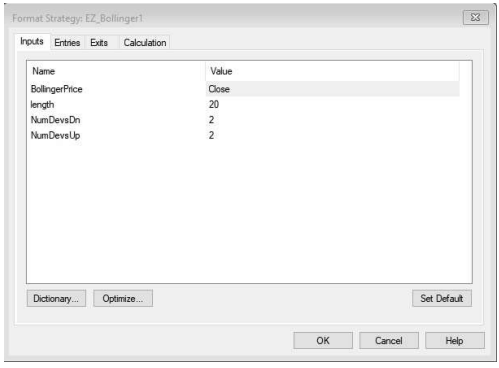
Now that we know a couple of the basic tenets of Easy-Language, let us move on to another indicator that is much utilized in the industry, **Bollinger Bands**. Bollinger Bands consist of three lines: the center band (moving average), the upper band (upper price channel) and the lower band (lower price channel). The upper and lower bands will move away from the moving average as price volatility increases and will move toward the moving average as volatility decreases. The default value for the moving average (MA) is 20 days, while the default value for the upper band (UB) is 2 standard deviations above the MA and the default value for the lower band (LB) is also 2 standard deviations below the MA. Let us insert the preinstalled TradeStation Bollinger Band indicator by creating a new chart of @CL, or a market of your choice, and then right clicking on your chart and choosing **Insert**

Analysis Technique. From this menu, under the Indicator tab, simply click on Bollinger Bands, hit OK and you will see the bands (assigned the default values of 20 (MA), 2 (UB) and -2 (LB). To double check, right click on the chart again, click **Format Analysis Technique** then click **Format** again, and you will see the inputs, which you can change at any time. Now let us incorporate this indicator into a new strategy. Make sure your TradeStation Development Environment, or as we will refer to it throughout the rest of this tutorial the TDE, is open. Create a new strategy and save it as EZ_Bollinger1. Type the following line of code into the editor and Verify it.

EZ_Bollinger1

```
Inputs: BollingerPrice(Close),length(20),NumDevsDn(2),
NumDevsUp(2);
```

This one line of code is a complete computer program, or in the vernacular of EasyLanguage, a complete strategy. It doesn't do anything, but it is completely valid.



Turn off the existing Donchian strategy if it is still turned on and apply this new strategy to your chart. When you click on **Format Strategies** then

Format, you will see the following:

You can do this if you want to make sure your **Inputs** are showing up properly. Now let us add a few variables – which are used to simply represent or hold a user defined value that you can incorporate later into your code. These can be expressed using letters and numbers. Variable names cannot be named solely using numbers, cannot start with a number and cannot include any mathematical notations.

Proper Variable Name Examples

Here is an example of some valid variable names:

- mySlowStochastic
- rsiValue1
- myDailyResult
- myFuncResult
- zzYYxx

The two important things you need to know about variables is that they are also considered **Bar Arrays or Lists** and once the variable is assigned a value it will continue with that same value throughout the back test until changed. Since we are looping through each bar from left to right in a chart, each variable can be indexed by a bar number offset. If you want to know what the **rsiValue** 10 bars back was, you can simply look at **rsiValue[10]**. The left square bracket number right square bracket [10] in-

forms the compiler that you want to reference the value 10 bars back from the current bar in the historic test. All the price based values such as **open**, **high**, **low**, **close**, **range**, **trueRange**, **volume**, and **openInt** (open interest in futures) are accessed in a similar fashion. Are you familiar with the term True Range? If not, remember range is the difference between the high of a bar and the low of a bar. True range equals the range unless the prior bar's close > the current bar's high or the close of the prior bar is less than the current bars low. So True Range is expanded to the prior day's close if it is outside today's range.

```
TrueRange = max(yesterday's close, today's high) -
      min(yesterday's close, today's low)
```

Gaps between the close and the following day's price extremes are incorporated in the True Range. True Range will always be equal to or greater than range.

If we want to know what the close was 5 bars back you can retrieve that information by simply stating:

Value1 = **close[5]**;

You always use a positive number inside the square brackets. A negative would suggest peeking into the future. If you set a variable (**myTempVal** or whatever name

you choose) to 2 it will be that value until you change it again. In some languages it would be up to you to reassign the variable the same value on the next bar. So EasyLanguage takes care of this for you. It is always best to use your own variable names for ease of readability, but sometimes you just need to quickly do something or you don't care about readability (I guarantee readability will eventually become important to you) and in these cases you can use EasyLanguage built-in variable names for numbers and conditions:

Value1 thru **Value99** can be used without declaring them in the **Vars**: section and be used just like any variable that you create yourself. **Condition1** thru **Condtion99** can also be used to hold True/False values.

Bollinger Band Based Algorithms

Type the following under your Inputs: section.

```
Variables: UpperBand(0), LowerBand(0);
```

Here we are telling the computer to reserve the space for two variables named **UpperBand** and **LowerBand**. At the same time, we are initializing the variables to the value of zero. You must always initialize your variables to some value. Most of the time you will just set them to zero by placing 0 in the parentheses that follow the vari-

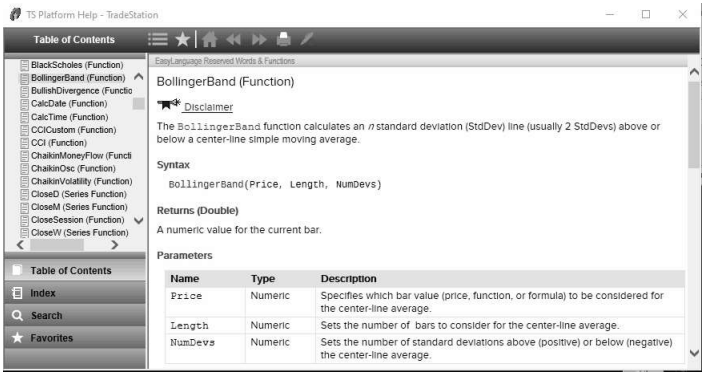
able names. All variables must be declared and initialized in a **Vars:** or **Variables:** statement (again notice the colon). You can initialize a variable with a number or a False or True value. If you use True or False, then the variable will become of the type Boolean. And this variable should only contain True or False values. Now let us use those variable names and assign the upper value of the **BollingerBand** function to **UpperBand** and the lower value of the Bollinger Band to **LowerBand**.

```
UpperBand =
BollingerBand(BollingerPrice,length,NumDevsUp);
LowerBand = BollingerBand(BollingerPrice,length,-Num-
DevsDn);
```

Notice how we use the BollingerBand functions and how we passed the input variables into the function? How did I know what the function needs (parameter inside the parentheses) to produce the correct calculation? This requires some research on your part. If you type Bollinger Band in the editor and then right click and select Definition of Bollinger Band you will get this dialog box. Notice how I used a minus "-" sign in front of **NumDevsDn?** The function expects a negative input for the **LowerBand.** You could allow the user, via the input to put a negative -2 in the input for the **LowerBand**, but that would require

them to know this information. The less information the user of the Analysis Technique needs to know the better.

Using TradeStation' Help With Indicators and Functions



Now copy those two lines of code and paste into the editor. By defining the variables, we can reduce redundancy in our coding and make it much easier to read. Let us now code a basic Bollinger system – under your variables type:

```
If c crosses over UpperBand then
begin
      Buy ("BBandLE") this bar on close;
end;
If c crosses under LowerBand then
begin
      SellShort ("BBandSE") this bar on close;
end;
```

This entry code directive states that if the close (you can simply use the letter "C" or "c" for the close price or the word "close") of the day crossed the **UpperBand**, then we will buy (user defined "**BBandLE**") at the close of that day. Alternatively, if the close of the day crosses below the **LowerBand**, then we will sell (user defined "**BBandSE**") at the close of the day. There are a few things going on here that I need to emphasize.

Price Data Shortcuts

The first letters of the words open, high, low, close and volume can be interchanged with the actual names: **O** for open, **H** for high, **L** for low, **C** for close and **V** for volume. You can use either upper or lower case.

The Cross(es) Keyword

The word Crosses is an immensely powerful function. It is a function that does not require any parameters to be passed, but must always have one of the following words follow it: **above, below, over** or **under**. **Crosses above** informs EasyLanguage that the closing price moved from below the **Upperband** to above the **UpperBand**.

```
Crosses over UpperBand translates into:

close[1]<UpperBand and close[0]>UpperBand
```

This does not simply mean the close is above the **UpperBand**; it means within the past two days the market's closing price crossed the **UpperBand** value (yesterday's close was *below* yesterday's **UpperBand** and today's close was *above* today's **UpperBand**). Crosses Below is just the opposite. Over is the same as Above and Under is the same as Below.

Conditional Branching with If-Then

This construct branches the path of execution of a program by testing a condition and if it is true then do something and if not do something else.

The **if-then** construct in EasyLanguage may or may not use the two following keywords.

- `Begin`
- `End`

This is remnant of Pascal. If only one statement is to be executed by the **if-then**, then a **Begin** and a matching **End** is not necessary. In the prior example we did not need to use the Begin and End, but they were included for illustrative purposes. Here is what the code would look like without the Begin and End. To ensure readability,

only us **Begin/End** if absolutely necessary. However, if you are unsure go ahead and use them.

```
If c crosses over UpperBand then
      Buy ("BBandLE") this bar on close;
If c crosses under LowerBand then
      SellShort ("BBandSE") this bar on close;
```

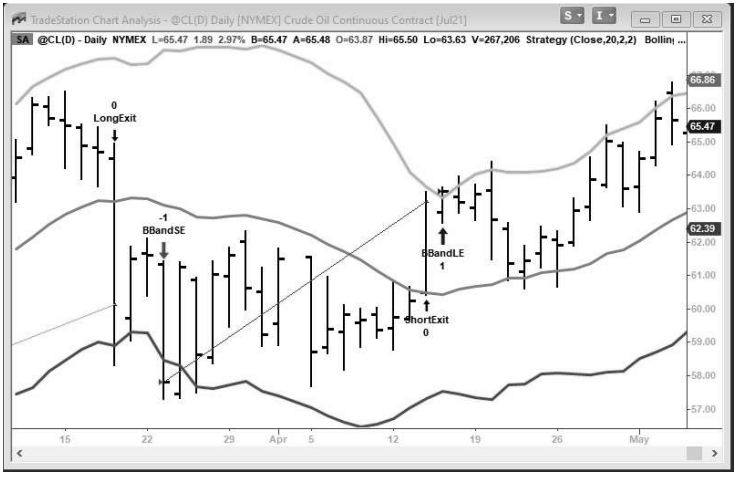
If more than one statement is to be executed by the result of the **if-then** you must use **Begin** and **End**. Indentation makes reading so much easier. After a Begin is used to signify the block of code that is to be used if the conditional test is true, an indentation by using the TAB helps the reader of the code understand that this particular block is controlled by the outcome of the **if-then** logical test. In other words, it is easier to see how the computer flows through the programmer's intent.

Since we are discussing **if-thens** and **begins** and **ends**, this might be a great place to introduce the **if-then-else** construct. This construct diverts the path of execution by providing an alternative to what occurs if the logical test fails. Here are the different permutations of how you can use the **if-then-else** construct. Again the **Begin** and **End** will be needed if you have more than one statement following the then or the else.

```
// quick and dirty
If Condition1 Then Value1=value2
Else Value1=value3;
```

```
// nice looking - easy readability
If Condition1 Then
      Value1=value2
Else
      Value1=value3;

//multiple statements after then and/or else
If Condition1 Then
Begin
      Value2=Value4;
      Value1=value2;
End
Else
Begin
      Value2=value3;
      Value1=Value4;
End;
```

The following chart illustrates our entries; every time the close is above the **UpperBand**, we buy, every time the close is below the **Lower Band**, we sell short. You will notice a **ShortExit** or short liquidation



on the chart. Some systems are indeed designed to be in the market perpetually, but most have some sort of exit. Let us explore that in our next snippet of code. Type in the following after the entry code:

```
Vars: CenterBand(0);
CenterBand = average(BollingerPrice,Length);
if marketPosition=1 then
begin
      If c crosses below CenterBand then
            sell ("LongExit") this bar on close;
end;
```

Here, we are defining a specific exit point so we could potentially limit losses. We are stating that if the market is long (if **marketPosition** = 1 (**long = 1, short = -1, flat = 0**)), we will sell when the close of the day is below the 20-day MA (CenterBand). If the market is short (if **marketPostion** = -1), we will buy when the close of the day crosses above the 20-day MA. The function call **average(BollingerPrice, length)** returns the 20-day moving average.

The prior chart indeed showed a short liquidation. Again, if we are long, we will sell when the close is below the 20-day MA, and if we are short, we will buy when the close is above the 20-day MA. Here is the code in its entirety:

```
//EZ_Bollinger1
```

```
inputs:BollingerPrice(Close),length(20), NumDevsDn(2),Num-
DevsUp(2);


variables: LowerBand(0),UpperBand(0),CenterBand(0);

LowerBand=BollingerBand(BollingerPrice,length, NumDevsDn);
UpperBand=BollingerBand(BollingerPrice,length,NumDevsUp);

CenterBand=Average(BollingerPrice,length);
If c crosses over UpperBand then
begin
      Buy ("BBandLE") this bar on close;
end;
If c crosses under LowerBand then
begin
      SellShort ("BBandSE") this bar on close;
end;
if marketPosition 1 then
begin
      If c crosses below CenterBand then
            sell(“LongExit”) this bar on close;
end;
if marketPosition 1 then
begin
      If c crosses above CenterBand then
            buyToCover(“ShortExit”) this bar on close;
end;
```

Notice to liquidate a long position you must use the **Sell** directive. **BuyToCover** is the keyword used to liquidate short positions.

Before looking at the results you might want to watch –

The video for Tutorial 3:

https://vimeo.com/578272185/2bfc2d3b38

Here are some results of this simple system:

TradeStation Performance Summary

	All Trades	Long Trades	Short Trades
Total Net Profit	$25,820.00	($26,530.00)	$52,350.00
Gross Profit	$183,830.00	$43,330.00	$140,500.00
Gross Loss	($158,010.00)	($69,860.00)	($88,150.00)
Profit Factor	1.16	0.62	1.59
Roll Over Credit	$0.00	$0.00	$0.00
Open Position P/L	$0.00	$0.00	$0.00
Select Total Net Profit	($14,870.00)	($26,530.00)	$11,660.00
Select Gross Profit	$143,140.00	$43,330.00	$99,810.00
Select Gross Loss	($158,010.00)	($69,860.00)	($88,150.00)
Select Profit Factor	0.91	0.62	1.13
Adjusted Total Net Pro	($27,944.57)	($50,920.06)	$1,131.09
Adjusted Gross Profit	$150,813.16	$32,142.24	$105,375.00
Adjusted Gross Loss	($178,757.73)	($83,062.30)	($104,243.91)
Adjusted Profit Factor	0.84	0.39	1.01
Total Number of Trades	89	43	46
Percent Profitable	34.83%	34.88%	34.78%
Winning Trades	31	15	16
Losing Trades	58	28	30
Even Trades	0	0	0
Avg. Trade Net Profit	$290.11	($616.98)	$1,138.04
Avg. Winning Trade	$5,930.00	$2,888.67	$8,781.25
Avg. Losing Trade	($2,724.31)	($2,495.00)	($2,938.33)
Ratio Avg. Win:Avg. Los	2.18	1.16	2.99
Largest Winning Trade	$40,690.00	$10,030.00	$40,690.00
Largest Losing Trade	($6,170.00)	($6,170.00)	($6,010.00)

Here is a streamlined version of the same code by eliminating the **Begins** and **Ends**. They were used deliberately to demonstrate their use.

```
//cleaner version
```

```
inputs:BollingerPrice(Close),length(20),NumDevsDn(2), Num-
DevsUp(2);

variables: LowerBand(0),UpperBand(0),CenterBand(0);

LowerBand=BollingerBand(BollingerPrice,length,-NumDevsDn);
UpperBand=BollingerBand(BollingerPrice,length,NumDevsUp);
CenterBand=Average(BollingerPrice,length);

If c crosses over UpperBand then
      Buy ("BBandLE") this bar on close;

If c crosses under LowerBand then
      SellShort ("BBandSE") this bar on close;

if marketPosition 1 then
      If c crosses below CenterBand then
            sell ("LongExit") this bar on close;

if marketPosition 1 then
      If c crosses above CenterBand then
            buyToCover ("ShortExit") this bar on close;
```

Definitely much more streamlined. Remember we were only able to do this, because in all cases, only a single line of code followed the **then** (this may not be evident due to the type of eBook reader being utilized). You can split a single line of code onto the subsequent line for ease of readability. It might look like. So,

```
If mp= then
```

```
      if c<c[1] then buy next bar at high limit;
```

is the same as:

```
If mp = 1 then
      if c  <c[1] then
            buy next bar at high limit;
```

The compiler parses each line until it reaches the line termination character – the semiColon (;). Comments are especially important when it comes to readability, but the commentary syntax is also highly useful for debugging purposes. EasyLanguage provides two methods to comment your programming code or temporarily toggle a line or lines of code on or off. We have already discussed the importance of readability and later on we will discuss the art of debugging. The first comment syntax is the double forward slash //. Do you know how to tell the difference between a forward slash versus a backslash? You can figure out which slash you are looking at by determining if it is tilting forward or backward. In this case, the slash is tilting forward. The code that **follows** the double forward slash will be ignored by the compiler.

```
// value1=99;
// programmed by George Pruitt
// version 2.1
```

Now if you have a huge block of code and don't want to have to put double forward slashes at the beginning of

each line, you can use **block comment** syntax and this is accomplished with the curly brackets **{ }**.

```
{Programmed by George Pruitt May 21, 2021
Vesion 2.1
Revision to determine days in long position
Added logic to exit after the eighth bar
of a long position}
```

Here we used an opening curly bracket and a closing curly bracket to surround the comment. I personally use both, but mostly the **{}** because I can comment the middle section of a line of code out. Take a look at this:

```
If condition1 {and condition2} and condition3 then
```

Condition2 is ignored by the compiler so only **condition1** and **condition3** need to be true for the test to pass. This is a peek to using the comment symbols to help debug your code. This little bit of code uncovered another huge topic – Boolean Algebra.

Boolean Algebra

Without Boolean Algebra, programming anything would be either impossible or exceedingly difficult. Fortunately, you do not need to know very much Boolean Algebra to accomplish converting you trading ideas into

source. Most of the time all you need to know is **AND** and **OR**.

"AND"ing occurs when you compare two or move values and both (all) must be true to return a true state:

```
If condition1=True and Condition2=False then
```

In this test you are testing to see if **condition1** is true and **condition2** is false. If either logical test fails then the whole expression fails or assumes a false state.

"Or"ing occurs when you compare two or more values and if any of the comparisons are true then a true state is established:

```
If condition1=True or
      condition2=False or
      condition3=False then
```

In this test you are testing to see if **condition1** is true or **condition2** is false or **condition3** is false. If any of the test are true then the whole expression passes and assumes a true state. If all are false then the test fails, and a false state is assumed.

You will be using "**And"** and "**Or"** a lot in you programming – this is another one of the things I can guarantee.

What Have We Learned From Tutorial 3?

- How to create and initialize variables for later use in our programs.
- Proper naming of variables – mix letters and numbers, but do not use strictly numbers or start variable names with a number. No math symbols either.
- Determining current market position with the **MarketPosition** function.
- How to use the **Crosses function** and how important it is.
- Utilizing **if** and **then** and their associated **begin** and **end**.
- Expanding program flow with the addition of the **Else** statement.
- Commenting code for readability and possibly debugging purposes
- **Applying Boolean Algebra** – just a little bit.
- Indentation, indentation, and indentation – how the cool programmers do it.

TUTORIAL 4: KELTNER CHANNEL INDICATOR UTILIZING INPUTS, DOLLAR STOPS AND DOLLAR PROFIT OBJECTIVES

Remarkably similar to **Bollinger Bands** in concept the **Keltner Channel** utilizes **Average True Range** (ATR) to generate channels above/below a moving average. As you already know the Bollinger Band indicator uses standard deviation to calculate its channels/bands. The ATR is a moving average of the **True Range** – usually defaulting to a period of 14 days. True range is defined as the highest of the following:

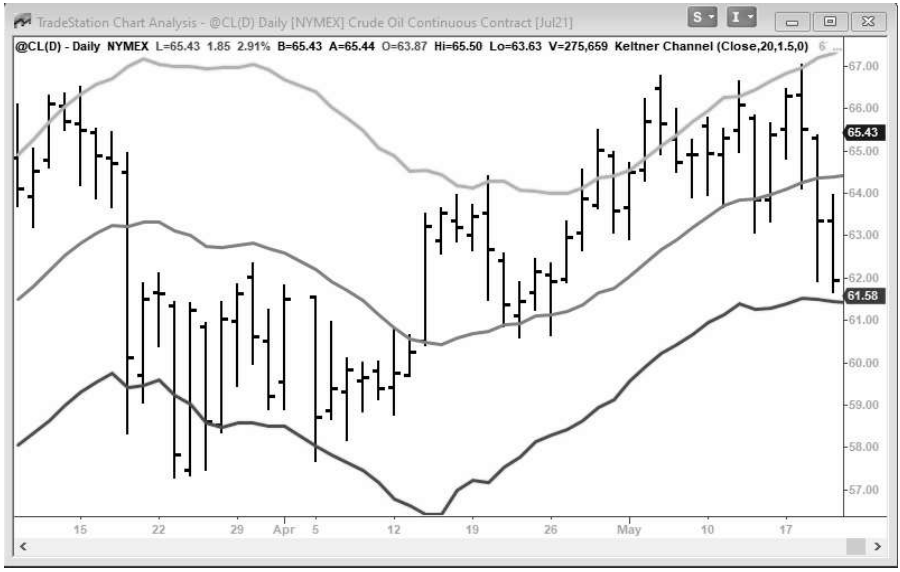
- The current high minus the current low
- Absolute value of the current high minus yesterday's close
- Absolute value of the current low minus yesterday's close

Like the Bollinger band indicator, the Keltner channel envelops a moving average and basically states that if the close of the day is higher than the upper band, then market is demonstrating robust momentum to the upside and a buy order should be executed, and if the close of the day

is lower than the lower band, then a sell short order should be executed. This is an amazingly simple trend following indicator that suggests a forthcoming bullish market if close exceeds the upper band and a bear market if it drops below the lower band.

Keltner Channel Based Algorithms

Similar to what we did with the Bollinger Band indicator, let us insert the preinstalled TradeStation Keltner Channel indicator by right clicking on your chart and choosing **Insert Analysis Technique**. From this menu, under the **Indicator** tab, simply click on Keltner Channel, hit OK and you will see the channels (usually assigned with the default values of 20 (Length) 1.5 (NumATRs)). To double check, right click on the chart again, click **Format Analysis Technique** then click **Format** again, and you will see the inputs, which you can change at any time. In the Crude Oil chart below, we are looking at a 20-day lookback with an ATR offset of 1.5.



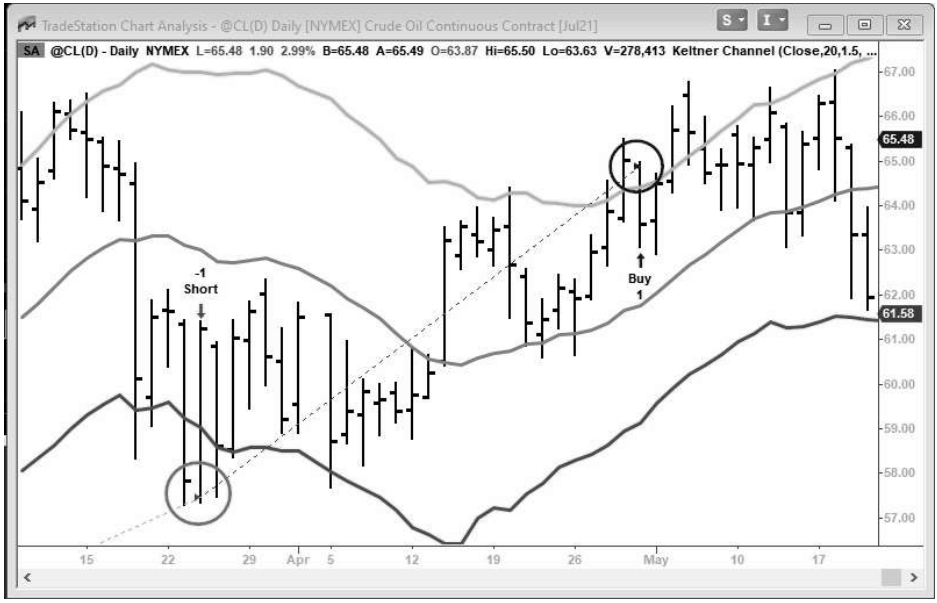
EZ_KeltnerBasic

Now we can add some basic code so we can generate signals. Go into your editor and type the following:

```
// EZ_KeltnerBasic
Inputs: keltnerLen(20),keltnerNumAtr(1.5);
if c crosses above
      keltnerChannel(close,keltnerLen,keltnerNumAtr ) then
            buy("KeltBuy") next bar at open;
if c crosses below
      keltnerChannel(close,keltnerLen,-keltnerNumAtr) then
                        sellshort("KeltShort") next bar at
open;
```

As you can see, we buy at the open if the prior day crosses above the upper band, and we sell short at the

open if the prior day crosses below the lower band.



EZ_Keltner1

In the prior scenario, we will always be in the market. However, let us suppose we wanted to set a stop order to limit our losses or a limit order to take a profit at a certain point. We can do this by adding the highlighted code below.

```
//EZ_Keltner1
inputs: keltnerLen (20), keltnerNumAtr (1.5),
profitAmt$ (1000), stopLossAmt$ (500);

if c crosses above
      keltnerChannel(close,keltnerLen,keltnerNumAtr) then
```

```
            buy("KeltBuy") next bar at open;
if c crosses below
      keltnerChannel(close,keltnerLen, keltnerNumAtr) then
            sellshort("KeltShort") next bar at open;
```

Incorporating a Profit Target and Protective Stop

```
if marketPosition 1 then
begin
   If c crosses below
      average(c, keltnerLen) then
         sell("LongLiq Avg") this bar on close;
      sell("longLiq") next bar
         at entryPrice-stopLossAmt$/ bigPointvalue stop;
      sell("longPrf") next bar
         at entryPrice + profitAmt$/bigPointValue limit;
end;

if marketPosition 1 then begin
   If c crosses above
      average(c,keltnerLen) then
         buyToCover("ShortLiq Avg") this bar on close;
      buyToCover("ShortLiq") next bar
         at entryPrice+stopLossAmt$/ bigPointvalue stop;
      buyToCover("ShortPrf") next bar
         at entryPrice-profitAmt$/bigPointValue limit;
end;
```

If the market penetrates the moving average that is integrated in the Keltner Channel indicator, then TradeStation is directed to get out of the trade. In addition, we are also defining specific dollar amounts with which

to exit the trade, in this instance, either with a $1000 profit or a $500 loss. These values can be changed by simply altering the inputs in your **Format Strategies** tab. If we are long, then we will sell ("**longLiq**") when our stop loss is hit, $500 from entry price or we will sell ("**longPrf**") when our profit target is hit, $1000 from entry price.

Stop and Limit Keywords

If we are short, then we will buy ("ShortLiq") when our stop loss is hit, or we will buy ("ShortPrf") when our profit target of $1000 is reached. You will notice that we must use the keywords **stop** and **limit** for these types of orders. Also, **next bar** must also be incorporated in the trade directive.

User Defined Profit Objectives/Stop Losses versus their Built-In Counterparts

I like to program my own profit objectives and protective stops by using if-then constructs and calculating the levels by dividing the dollar values by **bigPointValue**. **BigPointValue** is the constant for each futures/commodity market that equates to a full point move in the symbol that is being tested. In crude oil it is $1000. If a crude oil contract consists 1000 barrels and the current market price for crude is $62, then the notional value of 1 contract is 1000 * $62 or $62,000. If the price moves from $62 to $59 or $3, then that equates to 1000 * $3 or

$3,000. Using **bigPointValue** helps you to translate a dollar move to a price move. If your stop is $3000, then you can divide by **bigPointValue** to translate to a price move - $3. I like to program my own logic to exit trades, because I feel I have more control of the actual coding of the strategy, and may want to use some other stop or objective that might not be dollar or point based. But you do not need to go to these extra lengths if you are simply using dollar values or even point values (values derived by a calculation that results in points). EasyLanguage lives up to its name by allowing you to use these simple function calls:

- **SetStopLoss** – simply input the $ amount that want to risk
- **SetProfitTarget** – simply input the $ amount for you profit objective
- **SetStopPosition** - a switch that changes the behavior of the built-in stop command so that the exit order is calculated on a total position basis, verses a per share / per contract basis. Your strategy will default to this behavior unless you specify otherwise.
- **SetStopShare** – another switch that set the stop loss or set- Profit target on a single share or contract basis. You must insert this into your code for

this behavior to work. So instead of SetStop-Loss(500) you would state SetStopLoss(5) [five dollars per each share].

Here is a link to the video associated with Tutorial 4:

https://vimeo.com/578577571/2822d8820c

EZ_Keltner2

Here is an update to the code using these built-in functions.

```
//EZ_Keltner2

inputs: keltnerLen(20), keltnerNumAtr(1.5),profi-
tAmt$(1000), $stopLossAmt(500);

if c crosses above
      keltnerChannel(close,keltnerLen,keltnerNumAtr) then
            buy next bar at open;
if c crosses below
      keltnerChannel(close,keltnerLen, keltnerNumAtr) then

            sellshort next bar at open;

if marketPosition 1 then
begin
      If c crosses below average(c,keltnerLen) then
            sell this bar on close;
```

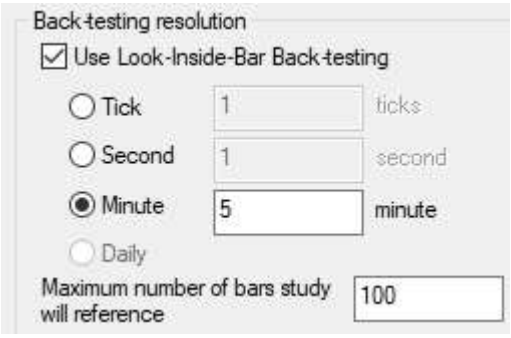
```
end;
if marketPosition 1 then
begin
      If c crosses above average(c,keltnerLen) then
            buyToCover this bar on close;
End;
setStopLoss(stopLossAmt$);
setProfitTarget(profitAmt$);
```

Another benefit of using these functions is that you can actually get out on the same bar you enter. If you program your own stop loss or profit objective similarly to what we did in the first version of the Keltner strategy, then you cannot get out until the following bar or subsequent bars after entry. So, if you are using tight stops or profit objectives and you want to get out on the bar of entry, then you wil want to make sure you use these functions. However, you will also want to turn on **LIB** [look inside bar].

Look-Inside-Bar [LIB] feature

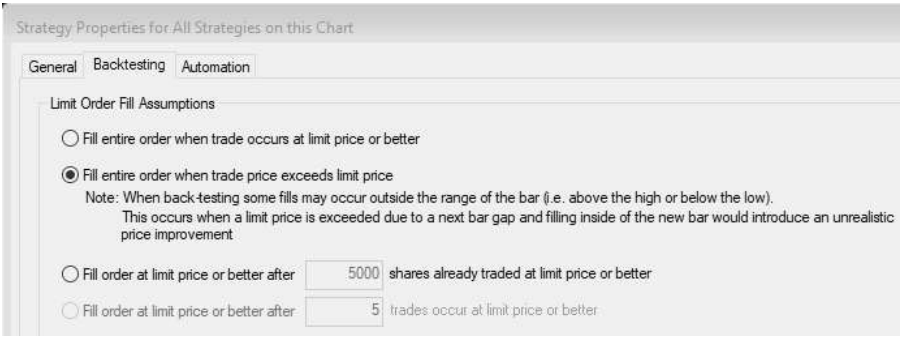
If you have the chance to either take a profit or get stopped out on the same bar, then you need to know what would have occurred first – the loss or the profit. If you turn on **Look Inside Bar** feature, and set the bar time interval to something appropriate to you trading algorithm, then TradeStation will pull intraday data in and determine if the market went in you favor or went the opposite

way first. If the initial move was large enough to either get you out of the market with a loss or a profit, then the sequence of these events are very important. Simply look at a daily bar and analyzing the relationship of the closing or opening price to the high or low is not always sufficient. Even if you do not use the built-in functions, but still have tight stops and profit objectives, you should turn this feature on.

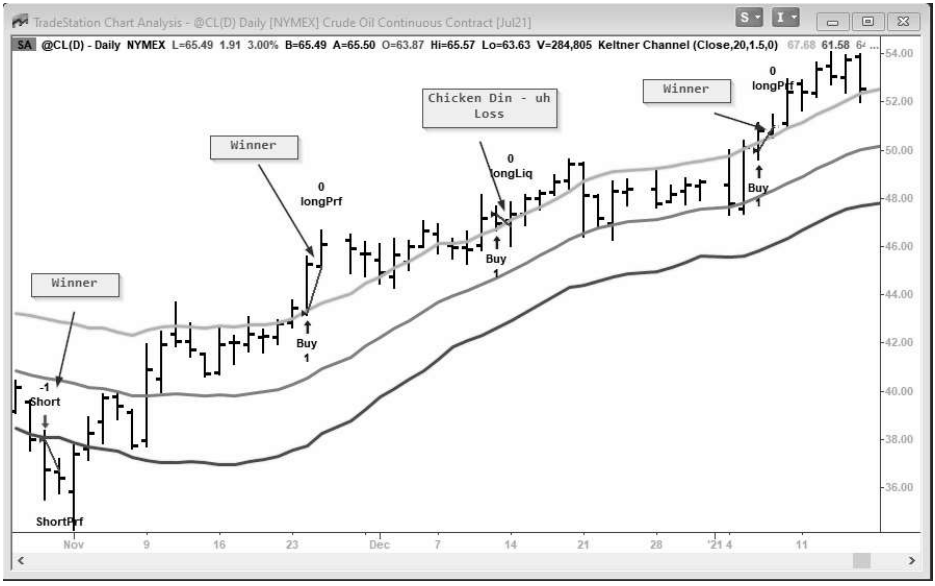


You can turn this on by formatting the Strategy and going to **Properties For All**. Another key feature to use when limit orders are applied to a strategy is this one:

Force Trade Through Price on Limit Orders



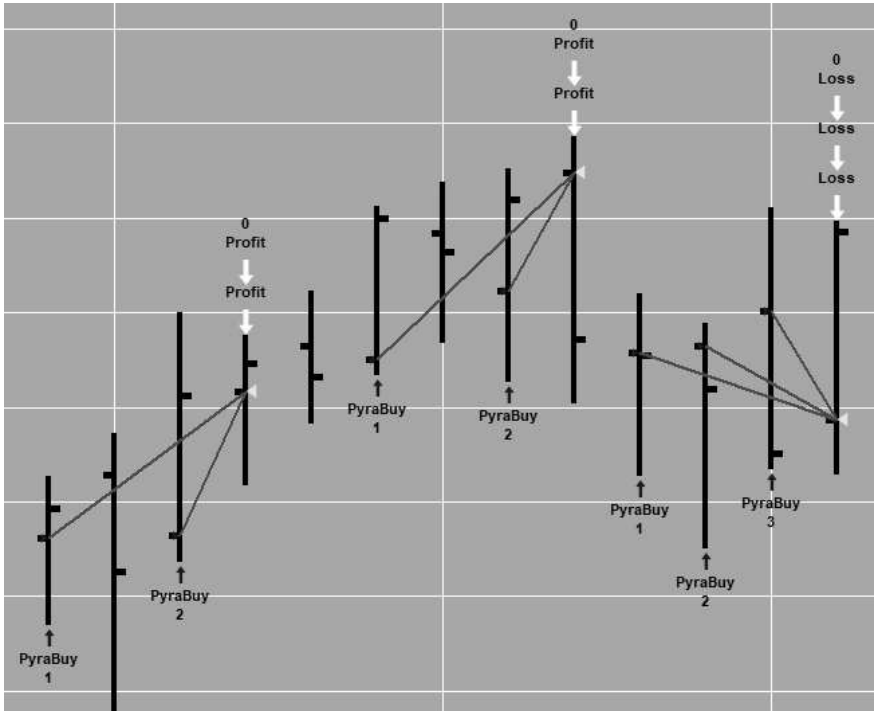
By selecting this option (Fill entire order when trade price exceeds limit price), you force the price to move completely through the limit price and guarantee that all positions are filled. If you do not select this, then all orders are filled on an "**if touched**" basis. I can't tell you the number of times my limit order wasn't filled at the limit price – even after tapping it several times.



What Have We Learned From Tutorial 4?

- In this tutorial we continued using inputs to adjust key variables in the decision process of when to enter long or enter short.
- Expanded on different exit mechanisms by incorporating fixed dollar stops and profit objectives.
- Explained the meaning of **BigPointValue**.
- **LIB – Look Inside Bar**. This option was introduced to ex- plained. It is important to use whenever an algorithm needs to know which occurred first on an intraday basis – works well with strategies that incorporate tight stops and profit object- ives.
- Only fill limit orders if price pushes completely through the limit order level. Many times, limit orders are not filled at the limit price. Don't kid yourself – set the back test up to be as difficult as possible.

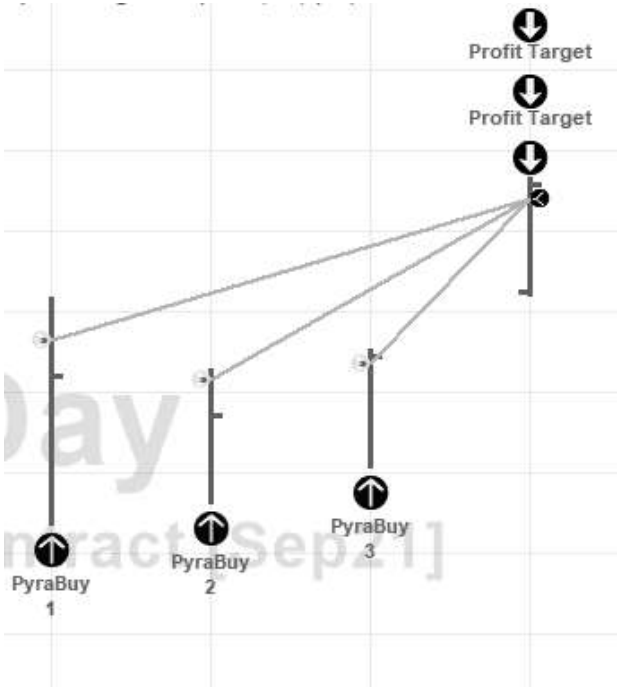
TUTORIAL 5: MEAN REVERSION AND PYRAMIDING



Now we are stepping on the accelerator. This might be a little advanced at this level, but all the information that you need to do this has already been explained. This is a tough chapter, but if you get through it, you will be well

on your way. I am going to show you how to pyramid up to 3 positions on a daily bar basis whenever the market moves away from a longer-term moving average.

The markets that we will be working with are the ES (mini SP) futures and Apple stock and the test period will cover the last 8-15 years. Here is a quick screenshot of how this should work – put on up to three positions and either take a $1500 profit or $750 loss based on the initial entry price.



Pyramid Algorithm Explained

Here trades are entered as long as the close is above the 100-day moving average and the close is less than the prior day's close. Looks good thus far, but things are not as EZ as we might first think. This is why we are EZing into EasyLanguage. Here is the simple code for this strategy.

EZ_Pyramid

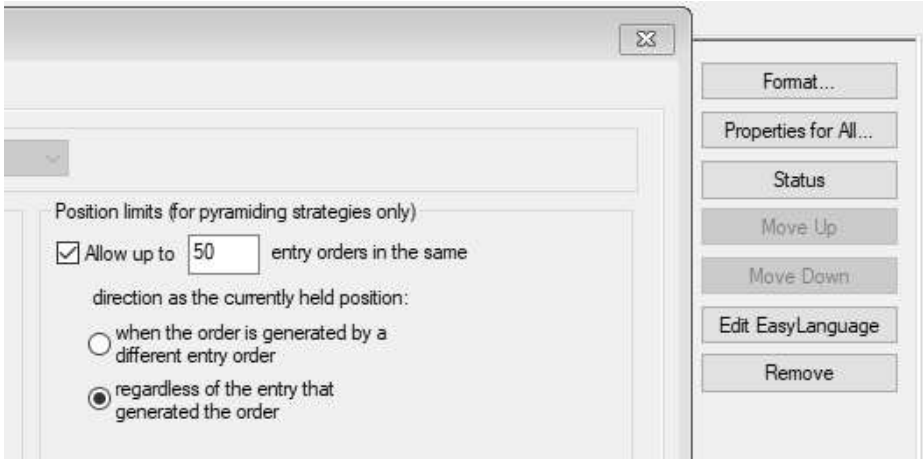
```
//EZ_Pyramid
input:trendLen(100),maxPositions(3),profitObj$(500),
maxLoss$(500);
Vars: mp(0);

Mp=marketPosition;
If close>average(c,trendLen) then
Begin
      If c<c[1] and currentShares<maxPositions then
            buy("PyraBuy") next bar at open;
end;
If mp=1 and c>entryPrice+profitObj$ /bigPointValue then
      sell("Profit") next bar at open;
If mp 1 and c<entryPrice-maxLoss$ /bigPointValue then
      sell("Loss") next bar at open;
```

Here the code makes perfect sense:

- Inputs are used to define variables we might want to change in the future. Notice when inputs are in terms of dollars the $ sign is used in the input name.
- Variables are setup and declared.
- If-then construct used correctly. Only take on a position if the close > 100-day moving average and the close is less than the prior day's close and the current number of shares is less than maxPositions.
- **CurrentShares** or **currentContracts** keywords are used to determine the current market position size.
- Stop Loss and Profit Objectives set accordingly.

Turn Pyramiding Option On

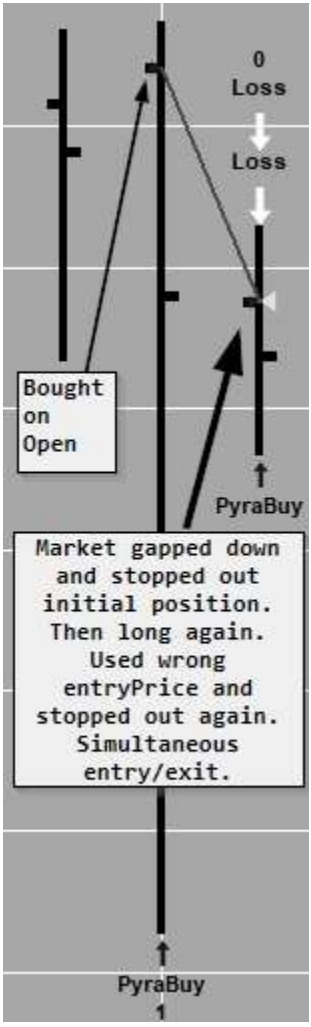
Now this program will not add on pyramid positions unless you format the strategy and select this option. You



can find it when you **Format the Strategy** and go under the **General** tab.

This basically tells TradeStation to add up to 50 positions in the same direction no matter what signal generated the order. This particular strategy is limiting the number of max shares or contracts to three. Like most programming, you do not know if your code is working like it should until you examine a bunch of trades. Here is an example of where it is not working.

Code Not Working Right



If you look at the graphic you will notice the strategy entered long, and then exited with a loss on the open and immediately went back long and stopped out again – two losses simultaneously. This is not what we programmed or is it? In reality what should have happened here.

1. Entered Long
2. Stopped out on entry bar for specified loss amount
3. Enter Long on subsequent bar

Yes it did exactly what we programmed it to do. How can that be, you might ask? Well, we told it to enter on the open after a down close and set an exit (either a profit or loss) at the **entryPrice** plus or minus a certain amount. Since we are not using the built-in stop loss function we cannot get out on the entry bar. Another thing about TradeStation that will make you pull your hair out is that **entryPrice** needs an intervening bar to be updated properly. Here we do not have one so **en-**

tryPrice is not updated and is still the same price as the first entry price even though the trade is long gone.

EntryPrice Needs an Intervening Bar to Update

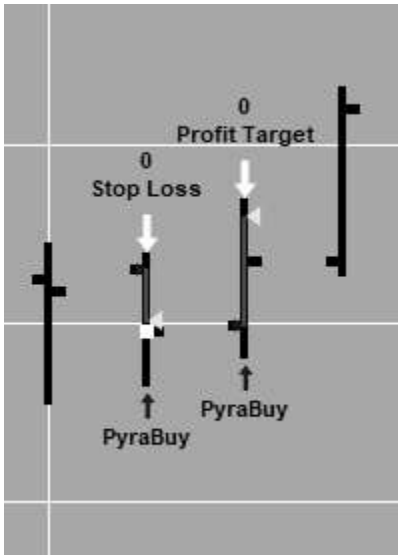
In this example TradeStation is using the wrong **entryPrice** price and subtracts the dollar amount, then we should be out at the open (simultaneous entry and exit). Can we overcome some of these limitations? Sure, let us ease into it. The major SNAFU is not getting out on the entry bar and using the wrong **entryPrice** to calculate the subsequent trade loss level. If you do this to the code:

```
//If mp=1 and //c>entryPrice+profitObj /bigPointValue then
//    sell("Profit") next bar at //open;
//If mp=1 and c<entryPrice-//maxLoss /bigPointValue then
//    sell("Loss") next bar at open;
SetStopLoss(maxLoss );
SetProfitTarget(profitObj );
```

Enable Exit On Entry Bar

Comment out the programming logic that issues the Profit and Loss trade directives and replace them with the bultin SetStopLoss and SetProfitTarget (put two forward leaning slashes at the beginning of each line of code). It turns out this sort of works. Take a look at these two trades and see for yourself (note to yourself – if you are

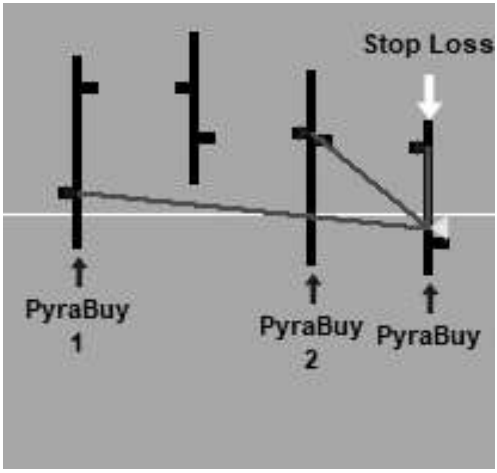
using close stops and profit objectives make sure you turn **LIB** on).



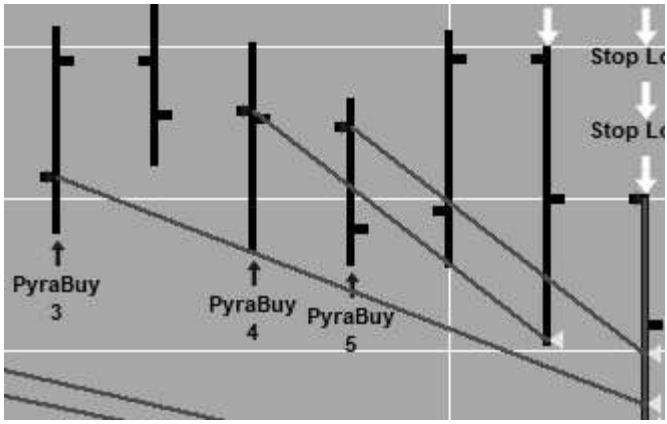
The first trade enters at the open of the bar and then later on gets stopped out (on the same bar as entry). The criteria are still set for another entry on the next bar's open, so it is executed and later on in the day the profit objective is reached.

Code Almost Working

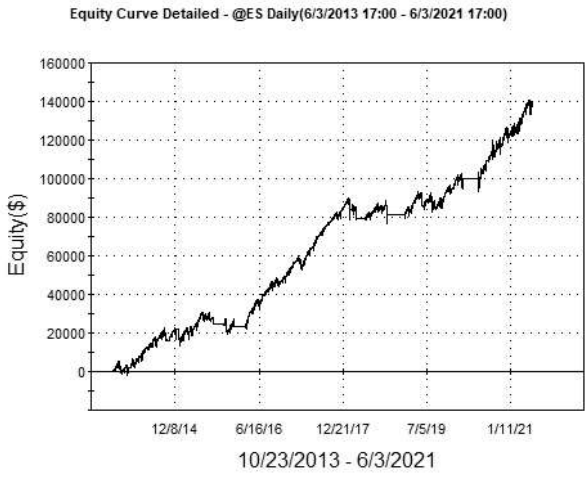
The next chart shows three entries and eventually a complete liquidation at a loss. In these examples $1500 was used as the objective, and $750 as the loss.



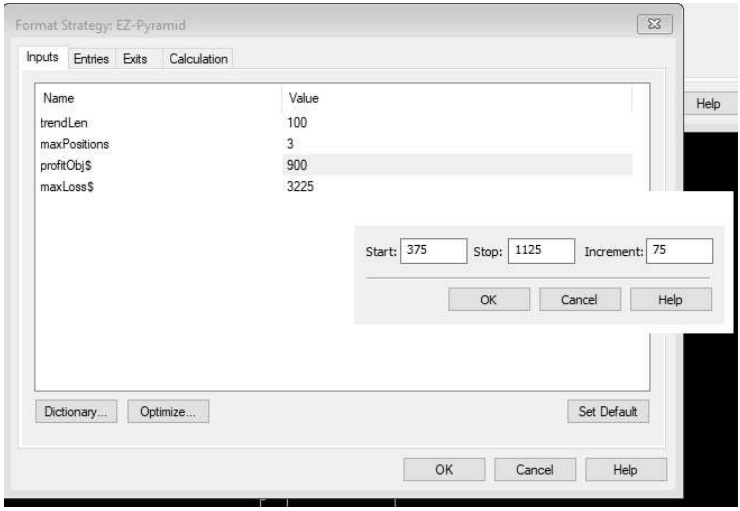
If you look closely, you will see the $750 is for the entire position, not $750 for each contract. What happens if we set **SetStopContract**? This will force each position or contract to risk individually $750.



That did not work that great either. If we want to exit at a $750 loss from the original position for all **addOns** and want entry and exits to occur on the same bar, then you will need to drop down to intraday data. This type of programming is much more complicated and is the basis for the second book in this series. The best way to program pyramiding and maintain a fixed level of risk, is to not use **SetStopContract**, and you will need to expand total position risk. So, if you want to place a stop for all positions at $750 from your very initial entry price then you will need to come up with an acceptable total position risk. With this design you are hoping to enter additional positions at better prices, but that does not always work as we have seen from the examples. Taking that into consideration you will need to come up with a new risk metric. I would suggest using a value less than or equal to 3 X $750 or $2250. Check this equity curve out.



Remember this strategy was almost guaranteed to work because it was buying pullbacks in a strong bull market, and this curve also reflects the optimal parameters for risk and reward. When the market entered into a somewhat bearish posture during the genesis of the pandemic, the algorithm basically went dormant - the trend filter took over. When developing an algorithm you can sort of guess what the risk/reward model should be, but this is where optimization can put you into the ball park. Using optimization to create a baseline is not curve fitting. Here is how you set up an optimization run for this particular algorithm.



I sort of knew what values to use, but was not 100% sure, so I did a quick optimization. Since daily bars was utilized this optimization took just a few seconds. And since only two variables were optimized the number of tests were constrained by the number of optimizations of **profitObj**$ times the number of optimizations of **max-Loss**$. Assuming 11 tests for each parameter, then 11 X 11 = 121 total runs. Now if you were to optimize a third parameter 11 times then the total number of tests would grow to 11 X 11 X 11 = 1331.

Here is the link for Tutorial 5 video:

https://vimeo.com/578721322/57953c9b2e

Optimization and Factorial Growth

This growth rate is factorial and will eventually, with a large set of objects, exceed exponential growth. So optimizing three or more variables across a modest search space requires a genetic optimization approach. Fortunately, we are not there yet. Here are the best values optimizing just the **profitObj**$ and **maxLoss**$.

	EZ-Pyramid: profitObj$	EZ-Pyramid: maxLoss$	Test	All: Net Profit	All: Max Intraday Drawdown	All: % Profitable	All: Avg Trade
1	900	3,150	101	139,687.50	-11,850.00	78.07	217.24
2	900	3,375	112	131,600.00	-12,700.00	78.38	204.67
3	1,050	2,925	91	130,837.50	-18,937.50	75.04	205.40
4	1,050	3,150	102	130,687.50	-19,600.00	75.67	205.16
5	900	2,925	90	125,725.00	-15,125.00	76.71	195.23
6	1,050	2,700	80	122,812.50	-17,475.00	73.78	192.80
7	1,050	3,375	113	119,900.00	-21,137.50	75.51	188.23
8	1,200	2,925	92	116,562.50	-26,637.50	72.21	182.99
9	1,050	2,475	69	114,075.00	-16,125.00	72.50	178.24
10	900	2,700	79	113,587.50	-14,837.50	75.47	176.38
11	1,200	3,150	103	108,925.00	-28,512.50	72.48	171.27
12	750	3,150	100	108,650.00	-14,737.50	77.92	168.97
13	1,650	2,925	95	106,112.50	-37,950.00	66.08	170.60
14	900	2,475	68	104,712.50	-13,037.50	74.23	161.59
15	1,200	2,700	81	103,600.00	-24,462.50	70.96	162.64
16	900	2,250	57	103,012.50	-11,237.50	73.15	158.97
17	1,650	3,150	106	102,500.00	-37,612.50	66.94	165.32
18	900	2,025	46	102,200.00	-11,162.50	72.69	157.72
19	1,050	2,250	58	102,112.50	-17,175.00	70.56	159.05
20	750	3,375	111	101,087.50	-14,000.00	78.23	157.21
21	1,050	2,025	47	99,187.50	-13,362.50	69.52	154.26
22	750	2,925	89	98,125.00	-13,812.50	76.71	152.37
23	1,650	2,700	84	95,875.00	-33,750.00	64.80	153.40
24	1,200	3,375	114	95,650.00	-28,075.00	72.33	150.39
25	1,650	3,375	117	94,650.00	-37,875.00	67.63	152.42
26	750	2,025	45	89,787.50	-10,012.50	74.07	138.56

It boils down to basically risking 3 to 1 to get this equity curve. Do not go crazy and think you have found the Holy Grail. All that we have done is programmed a pyramiding, mean reversion algorithm. And how can you go wrong using this type of algorithm in a strong bull market? Just like the one that started in 2009 and thus far in 2021. The Holy Grail aspect to this strategy is to know when to use it and that is not an easy thing to do. The whole objective of this lengthy tutorial is to demonstrate EasyLanguage's pyramiding capabilities and its limitations.

Scaling In And Out

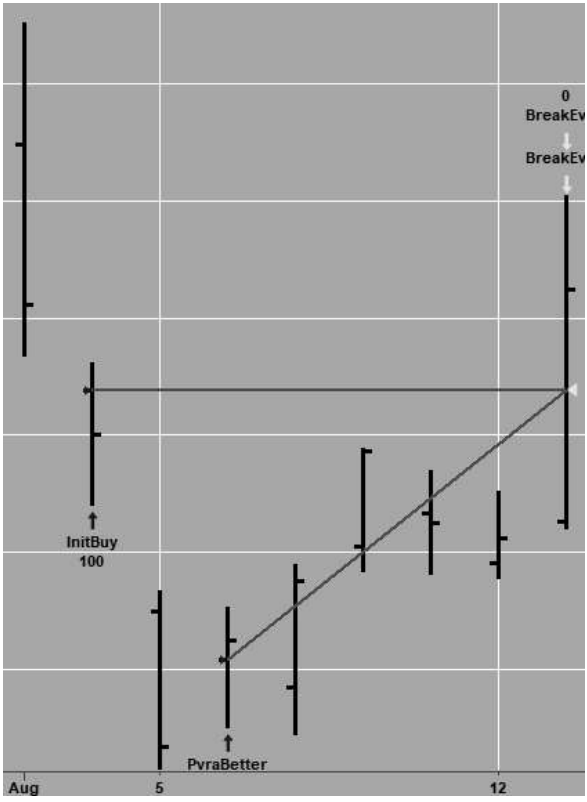
You can scale in and scale out using the reserved words **currentShares** or **currentContracts** and **total**. We might be jumping the gun a little bit here, but since we are thinking about pyramiding, let us continue in that vein.

This strategy will build upon what we have already, but let us add on some more logic.

- Add on 100 shares only at a better price
  - Add when price declines 1* averageTrueRange
- Move stop to original entry after adding on – break even
- Exit all positions after 20 days

Here is an example of a break even trade after putting on the additional 100 shares of AAPL. Here you see the

initial buy and then the market immediately falls more than one **Average True Range** (ATR) and then the next 100 shares are added on. If you



are not familiar with ATR it is a relatively easy calculation. **True Range** (TR) is the greater of today's high or yesterday's close minus the lesser of today's low or yesterday's close. Most times the True Range equals today's High – Low, unless you have a huge gap up or down. The ATR is then averaged over the last N days. The ATR is one of the most commonly used value that reflects market volatility. After the extra 100 shares are added, then the stop for the entire position is set equal to original entry price. This is considered a hybrid break even trade because you do make profit on the **add on** a.k.a. a free trade. This code is

adaptable to either futures or stocks. Here is the code and its associated inputs

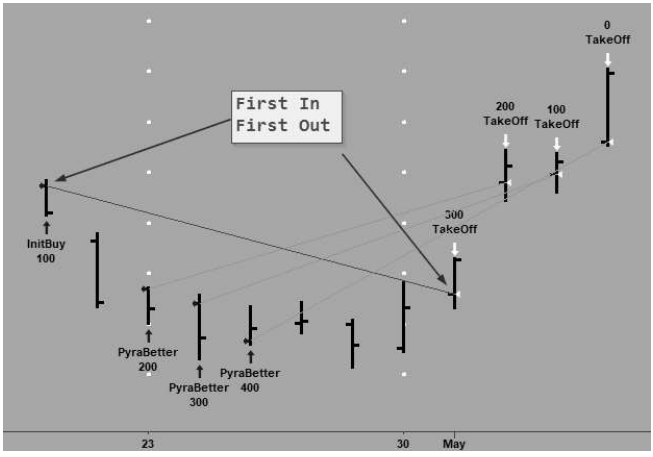
EZ_Pyramid2

```
//EZ_Pyramid2
input: trendLen(100), atrLen(10), maxPositions(2), profi-
tObj$(500), maxLoss$(500), numShares(100),
barsInTrade(20);

vars: mp(0);

mp = marketPosition;

If close > average(c,trendLen) then
Begin
      If mp = 0 and c < c[1] then
            buy("InitBuy") numShares shares next bar open;
end;

If mp = 1 and barsSinceEntry < barsInTrade and
      currentShares = numShares and
      c < entryPrice - avgTrueRange(atrLen) then
            buy("PyraBetter")numShares shares next bar
open;

If currentShares = numShares then
      SetStopLoss(Value1);
If currentShares = 2*numShares then
       sell("BreakEven") next bar at entryPrice limit;
If barsSinceEntry > barsInTrade then
      sell("BSESell") this bar on close;

Value1 = maxLoss$;
SetProfitTarget(profitObj$);
```

This code is similar to the original pyramiding code, but a few new keywords were introduced.

- **Shares** or **contracts** – this allows the programmer to set the size of the trade. As you can see an input is introduced numShares.
- **BarsSinceEntry** - number of bars that have completed since the initial entry.

The code that adds on the additional 100 shares (in this example) checks to make sure we are already long (mp = 1) and **barsSinceEntry** < 20 and the current close is an ATR less than the **entryPrice**. If these criteria are met then another 100 shares are added. If currentShares = 2 X numShares, then the computer knows that you have already added on the additional 100 shares and the stop must be moved to the original **entryPrice** ("BreakEven"). Once the trade has covered 20 days, then an order to close out all positions is issued ("BSESell") on this bar's close. This next portion of Tutorial 5 will introduce the concept of



scaling out. Here we will pyramid up to four (400 shares) positions at better prices and then start scaling out one position at a time. As the market starts to move down, positions are added 100 shares at a time (whenever the close is a multiple of ATR from the prior **entryPrice**) and then as it rebounds toward the mean, 100 shares will be peeled off. After adding the last 100 shares the market then rebounds 1/2 ATR from the last add-On entry. Logically, the last position should be the one that is popped off, but TradeStation uses a FIFO (first in - first out) scheme to scale out of positions. As the market continues to move up the second position is removed, then the third and finally the fourth. In this version, once the process of the initial buy and pyramiding has started all positions must be peeled off before a new initial buy and subsequent adds are allowed. This ought to be fun.

Scaling In and Out Code Breakdown.

Initial set up of inputs and variables. Obtain the current market position and then set up the criteria to issue the trade directive.

```
input:trendLen(100),atrLen(10),maxPositions(400),profitObj
(500), maxLoss (500),numShares(100),barsInTrade(20);

vars: mp(0),addOn(0),takeOff(0),ATR(0);
Mp = marketPosition;

If close>average(c,trendLen) then
```

```
Begin
      If mp=0 and c<c[1] then
            buy("InitBuy") numShares shares next bar open;
end;
```

So, if the market closes above the 100-day moving average and the current market position is flat and today's close < prior day's close, then buy numShares **shares** next bar at open. **Shares** is a keyword informing TradeStation to buy numShares (number of shares provided by user) shares at the next day's open. That is how you put on multiple positions or shares in an order directive - use the keyword shares or contracts.

Now how to add on shares at the appropriate prices?

```
If mp=0 then
begin
      addOn=0;
      takeOff=0;
      ATR=avgTrueRange(atrLen);
end;
If mp=1 and barsSinceEntry<barsInTrade and
currentShares<maxPositions and
currentShares=(addOn+1) * numShares and
C<entryPrice-(addOn+1) * ATR then
begin
      addOn=addOn+1;
      buy("PyraBetter") numShares shares next bar at open;
end;
```

The current market position must be long and **barsSinceEntry** must be less than the user's provided **barsInTrade** input and **currentShares** (another EasyLanguage keyword) is less than 1000. In addition, currentShares must be equal to (addOn + 1) * numShares and the close must be less than entryPrice - (addOn + 1) * ATR. The average true range ATR value is only captured when the market position is flat. Therefore, when adding on positions the same magnitude of price is constant. Let's slow down here. That is a lot of information to take in and how in the world did I come up with the math? I have programmed a ton of these approaches, so it comes second nature to me. But, let us walk through the logic.

We need a variable to keep track of the number of times we have added on (**addOn**.) Before you initiate a new long position, **addOn** (the variable) currently 0 is incremented by 1 so it will synchronize with the number of current shares that will be bought at the market (**addOn** -> 0 + 1 or 1 times **numShares**.) Since we haven't pyramided yet the current shares is equal to 1 times **numShares**. Has the close moved down below the entry price by (addOn -> 0 + 1) times ATR? If it has, then go ahead and add another 100 shares at the opening of the next bar. But before issuing the trade directive, go ahead and increment **addOn** (**addOn** = **addOn** + 1) again. We use this value as a multiple to let us keep track of the number of

stock shares that are currently on and calculate the next better buy level.

Add On and Take Off Math

If you go back through this logic with **addOn** set to 1 you get.

numShares calculation

addOn = 1

numShares = (addOn + 1) * numShares or

(1 + 1) * 100 = 200

next pyramid price level

addOn = 1

entryPrice = 1500

ATR = avgTrueRange(atrLen)

ATR = 150

next entry level = entryPrice - (addOn + 1) * ATR or

1500 - ((1 + 1) * 150) = 1200

From these calculations you can see how the number of shares stay in synch with the trade entry level. Now this next bit of code takes care of getting out after the market has bounced back up 1/2 ATR after the last long entry.

```
ATR=avgTrueRange(atrLen);
If mp = 1 and addOn > 0 and
C > entryPrice-addOn * ATR+(takeOff+1) * ATR/2 then
Begin
```

```
      Sell("TakeOff") numShares shares total next bar
open;
      takeOff=takeOff+1;
end;
```

Here another variable (**takeOff**) was created to help keep track of the positions that are scaled or peeled off and what level the scaling out/peeling off takes place. Using the same example and assuming 400 shares currently on, then this is the math involved in getting out.

last entry price = entryPrice - addOn * ATR

addOn = 3 (remember we just added 3 times)

numShares = (addOn + 1) * numShares

or

(3 + 1) * 100 =400

first scale out price level

takeOff = 0

addOn = 3

entryPrice = 1500

ATR = 150

last entryPrice = entryPrice - addOn * ATR

1500 - 3 * 150 = 1050

last entryPrice + (takeOff + 1) * ATR/2

scale out at 1050 + (0 + 1) *150/2 = 1125

next pyramid price level

takeOff = 1

scale out at 1050 + (takeOff + 1) * ATR/2

1050 + (1 + 1) * 75 or 1200

If you don't use the keyword **total** in the trade directive then the entire position will be liquidated. So make sure if you are scaling out then use the word **total**. Once all of the positions are liquidated (**addOn** = **takeOff**) the market position (mp) becomes zero and the process starts all over once again. You might ask why doesn't the logic add on after one or two initial scale outs? In other words, what prevents adding positions during the scaling out process? This is the logic that prevents that from happening.

```
If mp=1 and blah blah blah and
      currentShares=(addOn+1)* numShares
```

Once you start peeling shares off then **currentShares** no longer equals (**addOn** + 1) * **numShares**. This one portion of the expression keeps the **addOn** trade directive from occurring.

Could a beginner figure this code out for themselves? Sure, but it could take some time. But that is why you bought this book.

What Have We Learned From Tutorial 5?

This was a big tutorial and a lot was introduced and discussed.

- Pyramiding by using the option to allow multiple positions in the same direction.
- CurrentShares or CurrentContracts are used to determine the current position held.
- The keyword Total was used to peel off partial positions when scaling out.
- Shares or contracts – this allows the programmer to set the size of the trade.
- Use EntryPrice to determine the price of the initial position and simple formulas to calculate next entry size and price.
- BarsSinceEntry was used to calculate the number of bars that have completed since the initial entry.

TUTORIAL 6: USING MONEY MANAGEMENT IN YOUR TRADING ALGORITHM

In futures trading, testing on a one lot basis is very common. With TradeStation (not Maestro) it is usually easier to compare apples to apples using just one contract. That really is not an accurate statement. If you compare the current crude contract size to the mini-ES or a soybean contract you will see what I mean.

```
ES-BigPointValue=50
Notional Value=50 * 4220 (current price )=211,000

Cl-BigPointValue=1000
Notional Value=1000 * 72=72,000

Soybeans-BigPointValue=50 * 1543.25=77
Notional Value=50 * 1543.25=77,162.5
```

If you want to double check your math you can multiple the current price of a bushel of beans which is $15.4325 * 5000 bushels -> 5000 * 15.4325 = $77,162.5

Normalize Risk Among Different Futures

A **Fixed Fractional** approach is the most simple and utilized method to normalize risk among diverse futures contracts. Here is the basic formula:

```
Stake 100,000

Acceptable risk (AR):  2% of stake or $2000

Perceived Market Risk(PMR):  30 Day Average True Range
      PMR = $750

# of contracts (NUM) = AR / PMR
      or $2000/ $750 or 2 contracts (always round down)
```

Here is the code to manipulate the number of contracts.

```
//MoneyManager
Inputs: initCapital(100000),rskAmt(.02); Vars: market-
Risk(0), numContracts(0);

//two methods follow to calculate perceived market risk
//first is commented out
//marketRisk = StdDev(Close,30) * BigPointValue;
marketRisk=avgTrueRange(30) * BigPointValue;
numContracts=(initCapital * rskAmt) / marketRisk;

Value1=round(numContracts,0);
{Round down to the nearest whole number}
if(value1 > numContracts)
Then
      numContracts=value1-1
```

```
else
      numContracts = value1;
numContracts = MaxList(numContracts,1);

Buy("MMBuy") numContracts shares tomorrow at
      Highest(High,40) stop;
SellShort("MMSell") numContracts shares tomorrow at
      Lowest(Low,40) stop;

if(MarketPosition = 1) then
      Sell("LongLiq")next bar at Lowest(Low,20) stop;
if(MarketPosition = -1) then
      BuyToCover("ShortLiq") next bar at High-
est(High,20)stop;
```

This code introduces the Fixed Fractional approach to capital allocation. NumContracts is calculated by taking the initCapital($100,000) and multiplying it by rskAmt(0.02 or 2 %) to determine AR. In this example two methods are used to calculated perceived market risk (PMR): **AvgTrueRange**(30) and **StdDev**(C,30) (standard deviation). The **StdDev** is commented out by two forward slashes. Remember if you are using dollars you must convert a change in price to dollars as well. So the ATR is multiplied by **BigPointValue**.

Once the numContracts is calculated by dividing AR by PMR or

```
numContracts = (initCapital * rskAmt) /marketRisk;
```

it is then rounded to the nearest whole number by using the **ROUND** function.

```
value1 = round(numContracts,0);
```

Now the **ROUND** function will either round up or round down based on the value of numContracts. To make sure it rounds down you can use this logic.

```
if(value1 > numContracts) then
      numContracts = value1 - 1
else
      numContracts = value1;
```

At this point you should see what is going on here. The **if-then** construct is used to determine if the rounded (up or down) is greater than Value1. And if it is, then we know it was rounded up, so 1 is subtracted from Value1. If the two numbers are the same nothing is done to Value1. NumContracts is then assigned to Value1. If you want to make sure at least one contract is used then you can do this.

```
numContracts = MaxList(numContracts,1);
```

MaxList is another function call that extracts the largest value from a list of expressions, variables or constants. You will see the familiar **shares** keyword in the trade directives.

```
Buy("MMBuy") numContracts shares tomorrow at
      Highest(High,40) stop;
SellShort("MMSell") numContracts shares tomorrow at
      Lowest(Low,40) stop;
```

Here is a listing of the last two trades - notice how the 8/17 trade initiated three lots of beans and the 4/20 trade just two.

8/17/2020	MMBuy	$881.000	$0.00	3	$70,425.00
3/30/2021	LongLiq	$1,350.500		$70,425.00	$100,862.50
4/20/2021	MMBuy	$1,429.750	$0.00	2	$6,075.00
5/26/2021	LongLiq	$1,490.500		$6,075.00	$106,937.50

Compounding Profits

Now if you want to use compounding (reinvestment of profits) in your numContracts calculation then you will need to know, on a daily basis, what the amount of profits are. You can access this information through the keyword, **netProfit**. If you add **netProfit** to **initCapital**, then you can monitor equity growth and use it in your calculations. Here is the complete code - see if it makes sense to you.

```
//MoneyManger2 [LegacyColorValue    true];

Inputs: initCapital(100000),rskAmt(.02),maxSize(10);
Vars: marketRisk(0),numContracts(0),workingCapital(0);

marketRisk = StdDev(Close,30)*BigPointValue;
```

```
workingCapital = netProfit + initCapital;
numContracts = (workingCapital * rskAmt) /marketRisk;
Value1 = round(numContracts,0);

{Round down to the nearest whole number}
if(value1 > numContracts) then
      numContracts = value1 - 1
else
      numContracts = value1;

numContracts = MaxList(numContracts,1);
numContracts = Buy("MMBuy") numContracts shares tomorrow
at
      Highest(High,40) stop;
SellShort("MMSell") numContracts shares tomorrow at
      Lowest(Low,40) stop;

if(MarketPosition = 1) then
      Sell("LongLiq")next bar at Lowest(Low,20) stop;

if(MarketPosition = -1) then
      BuyToCover("ShortLiq") next bar at
            Highest(High,20)stop;
```

Here the only code that changes is the introduction of the variable workingCapital. WorkingCapital changes every time a trade is closed out. Since workingCapital is used in the numContracts calculation, you know profits or losses are used to calculate a new value. Remember how we constrained the numContracts to be at least 1,

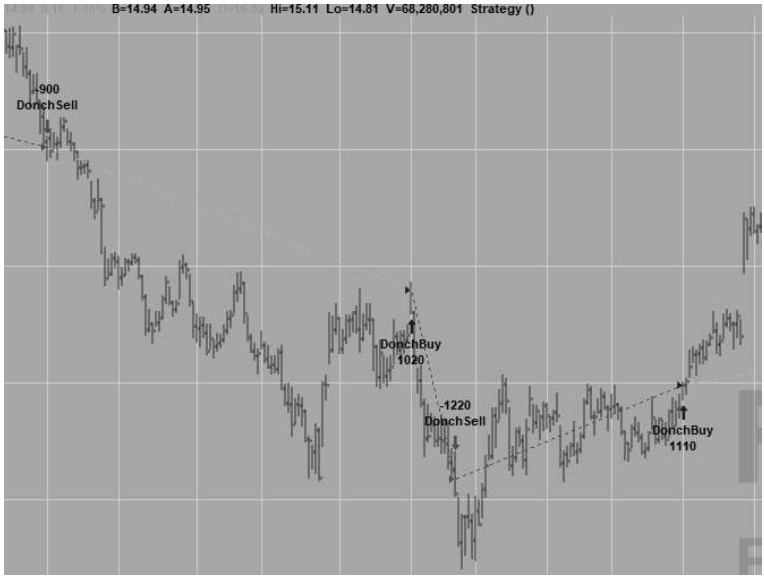
here we also constrain the numContracts to be at most a maxSize.

```
numContracts = MaxList(numContracts,1);
numContracts = MinList(numContracts,maxSize);
```

This time the function **MinList** is used. In this case, the smallest value is returned from the list. So we don't want the position size to exceed the maxSize. Compounding profits didn't make a huge difference in our test run. but four lots were initiated on the 8/17 and 4/20 trades versus three and two respectively.

8/17/2020	MMBuy	$881.000	$0.00	4	$93,900.00
3/30/2021	LongLiq	$1,350.500		$93,900.00	$108,687.50
4/20/2021	MMBuy	$1,429.750	$0.00	4	$12,150.00
5/26/2021	LongLiq	$1,490.500		$12,150.00	$120,837.50

Fixed fractional allocation is by far the simplest money management technique to use and it does a fairly good job of distributing risk across different markets. You can use this with stocks as well. If you want to allocate a fixed dollar amount when trading stocks you can simply pick a Strategy and via the Format Strategy:Properties.



You can choose USD per trade. The number of shares will then be calculated by dividing the dollar allocation by the price of the stock. Here is an example of allocating $10,000 on each trade in Ford (F).

Here is the link to the video for Tutorial 6:

https://vimeo.com/579520861/f3649563e2

What Have We Learned From Tutorial 6?

This tutorial explained why it is easier to test Futures on a one contract basis, but introduced how to use a Fixed Fractional approach to normalize risk. A simple $ allocation was shown for trading stocks

- Different Futures/Commodities can have totally different notional values. If you want to quickly prototype a strategy, a one contract approach is fine, but if you want to see a money management schemes that normalizes risk then you can't really go wrong with a Fixed Fractional approach.
- Convert all risk values to dollars to properly execute the formula. PMR is perceived market risk and can be measured by calculating the average true range.
- A pure dollar allocation can be achieved by changing the **Trade Size** option in **Format Strategy:Properties For All** dialog, if not specified within the Strategy itself.

TUTORIAL 7: CONNECTING SPECIFIC EXITS WITH SPECIFIC ENTRIES

Sometimes you want to use specific exits with specific entries. Assume you have two buy entries in one strategy and you have two exits and you want to tie those entries to their respective exits. This tutorial will show you how to do this.

Simple Strategy With Distinct Entries And Exits

This strategy will enter long on the first trading day of the month if the last day of the month's close is down and the close is greater than the 100 day moving average. The exit for this entry will occur on the third following day if a profit or stop loss is not elected first. Another long entry will occur after a two bar strength pivot high occurs. Pivot Points (High/Low), also known as Bar Count Reversals, are used to anticipate potential price re-

versals. Pivot Point Highs are determined by the number of bars with lower highs on either side of a Pivot Point High. Pivot Point Lows are determined by the number of bars with higher lows on either side of a Pivot Point Low.

For example, a Pivot Point High, with a period of 2, requires a minimum of 3 bars to be considered a valid Pivot Point. A minimum of 2 bars before and after the Pivot Point High all have to have lower highs.

At this point you are probably getting pretty good with EasyLanguage so let's just jump right in.

Exit From Entry Code

Remember you may not be able to copy and paste from the Kindle format and paste into your TDE.

```
//EZ_ExitLinkedEntry
inputs:sys1Prof (2000),sys1Loss (1500), sys1TradeDura-
tion(3), sys2Prof (1000), sys2Loss (750);

vars: mp(0);

// modularity is introduced with this algo
mp=marketPosition;

// entries are programmed here
If  dayOfMonth(date of tomorrow)<dayOfMonth(date) and
      Close < close[1] then
            buy("FirstDBuy") next bar at open;
```

```
If h[2] = swingHigh(1,h,2,10) and
      close > average(c,100) and
      close  < close[1] then
            buy("SwingHi[2]") next bar at open;

//exits are programmed here
If mp=1 then
begin
      If barsSinceEntry = sys1TradeDuration then
            sell("FirstDBuyXit") from entry("FirstDBuy")
            next bar at open;
      sell("FirstDBuyProf") from entry("FirstDBuy") next
bar
            at entryPrice + sys1Prof /bigPointValue limit;

      sell("FirstDBuyLoss") from entry("FirstDBuy") next
bar
            at entryPrice - sys1Loss /bigPointValue stop;

      sell("SwingProf") from entry("SwingHi[2]") next bar
            at entryPrice + sys2Prof /bigPointValue limit;
      sell("SwingLoss") from entry("SwingHi[2]") next bar
            at entryPrice - sys2Loss /bigPointValue stop;
end;
```

As you can tell we are getting a little more sophisticated as we go along. What do you think this is doing?

```
If  dayOfMonth(date of tomorrow) < dayOfMonth(date) and
close < close[1] then
      buy("FirstDBuy") next bar at open;
```

Here a new function, **dayOfMonth**, is introduced. In addition, **date of tomorrow**, is used as an argument (input) into the function. Remember when we discussed Future Leak and how TradeStation tries to eliminate it, but it allows you to sneak a peek at tomorrows open tick, well it also allows you to see the date of tomorrow. **DayOfMonth** returns the day of the month, if today is January 3, 2022, the function would return 3. If tomorrow's day of month is less than today's day of month what does that tell us? If you think about it, it tells us tomorrow is the first day of a new month. If so an order to buy on tomorrow's opening tick is issued as long as we have a down close. It is vitally important to give your signals names and in this case this order directive is labeled "FirstD-Buy". The other long entry code is.

```
If h[2] = swingHigh(1,h,2,10) and close          aver-
age(c,100) and close < close[1] then
      buy("SwingHi[2]") next bar at open;
```

This block of code contains compound comparison expressions and initially it might seem a little daunting. Also you are probably not familiar with the **swingHigh** EasyLanguage function. This function is really a time saver as it finds historic high pivot points of a certain strength and returns the pivot price.

```
h[2] = swingHigh(1,h,2,10)
```

This bit of code first instructs TradeStation to find the most recent high pivot with at least two lower highs before and after within the past 10 bars. If **h[2]**, high of prior day to yesterday, is that pivot price and the close is above the 100 day moving average (trend) and today's close is down, then buy the next bar. Basically a small pullback in the direction of the trend. Here this signal is labeled "SwingHigh[2]".

There are two distinct entry techniques - one based on the calendar day of the month and the other on a pattern. In cases where you have different entry schemes in the same strategy it is usually a good idea to marry the entry techniques to appropriate types of exits. This strategy has different profit objectives and protective stops based on whichever entry is elected. In addition, the "FirstDBuy" entry signal has an additional exit technique based on the number of days the trade has been in effect. This union between entry and exit is created by using the keywords **from Entry**("FirstDBuy"). Using this sequence of keywords and the name of the entry forces TradeStation to only apply the exit logic to the entry that is specified by the name inside the **from Entry** function, in this case ("FirstDBuy"). Here is the "FirstDBuy" exit module again.

```
If barsSinceEntry=sys1TradeDuration then
```

```
      sell("FirstDBuyXit") from entry("FirstDBuy") next
bar
            at open;
sell("FirstDBuyProf") from entry("FirstDBuy") next bar
      at entryPrice + sys1Prof /bigPointValue limit;
sell("FirstDBuyLoss") from entry("FirstDBuy") next bar
      at entryPrice-sys1Loss /bigPointValue stop;
```

The first exit logic looks to see if the trade has been in effect for **sys1TradeDuration** (User Input) days and if so it exits From Entry("FirstDBuy") on the next bar. The next two exit directives use specific profit (**sys1Prof$**) and loss (**sys1Loss$**) objectives. These exit directives will only be applied to the "FirstDBuy" entry.

```
sell("SwingProf") from entry("SwingHi[2]") next bar
      at entryPrice + sys2Prof /bigPointValue limit;
sell("SwingLoss") from entry("SwingHi[2]") next bar
      at entryPrice – sys2Loss /bigPointValue stop;
```

This module works with the "SwingHi[2]" entry mechanism. By using different profit and loss objectives for each entry technique you can optimize the values. When you optimize to many variables it is easy to fall into the trap of over curve fitting (bind- ing your algorithm to match historic data and therefore limiting flexibility.) However, since each entry technique is so different, it should be okay, but don't go nuts.

Here is the video link for Tutorial 7:

https://vimeo.com/580064016/8e6b1bc7a7

What Did Tutorial 7 Teach Us?

This tutorial explained how to tie a specific entry to a specific exit. Many times your algorithm will have multiple entries and exits and its important that the relationships between entries and exits make sense. You wouldn't exit a trader after one day if using a longer term Bollinber Band break out entry.

- Make sure you name you entry directives and make sure they have names that make sense.
- **From Entry**("EntryName") is the keywords and psuedo like function call that enables the container logic to only be applied to "EntryName".
- Connecting Entries and Exits allows for different algorithms to be included in the same strategy

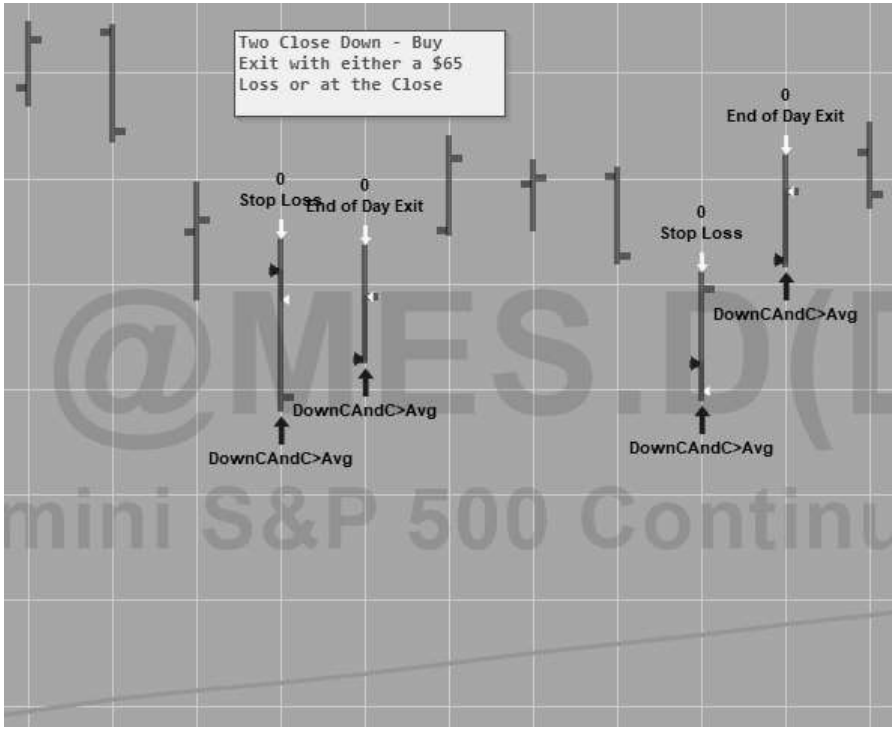
TUTORIAL 8: PROTOTYPING A DAYTRADE STRATEGY WITH DAILY BARS AND LIB (LOOK INSIDE BAR)

Do you have a day-trading idea but don't know how to program it at the intraday level (my next book in the series)? There are certain circumstances where you can test a day trading system with daily bars with and without the use of **Look Inside Bar** (refer back to Tutorial 4 for review.) When can you use a daily bar to test a day trading system without LIB?

Entry on the open with either a profit objective or protective stop, but not both. Why? Whenever you need to know what occurred first, the high of a move or the low, then you cannot test accurately with a daily bar. If you enter on the open and exit on the close and incorporate a protective stop the only thing the computer needs to know is if the low dropped below the open enough to stop you out. If you use a profit objective, the opposite is true;

the computer only needs to know the distance between the high and the open. Here is a chart using this strategy on the micro ES using parameters of 2 down closes prior to entry and close above 100 day average and a $65 stop.

Daytrader Prototype With Daily Bars



Here Is The Code

```
// EZ_DailyDayTraderCorrect
inputs: movAvgLen(100),numDownCloses(3),stopLoss (500);

If numDownCloses = countIf(Close-Close[1],numDownCloses)
and   C > average(c,movAvgLen) then
            buy("DownCAndC Avg") next bar at open;
```

```
setStopLoss(stopLoss );

setExitOnClose;
```

I try to introduce new functions and new ways to program strategies in each tutorial. In this tutorial the **countIf** function is used and so is the **SetExitOnClose**. CountIf is a function that returns the number of occurrences that a Boolean comparison occurs over a certain period of time.

countIf(Close < Close[1],numDownCloses)

This function requires a comparison and the number of this comparison is true of a period of time. The comparison, in this case, is a close less than a prior close and the span of time is **numDownCloses** or 2 days. So if **numDownCloses** (2) = the result of the **CountIf** function and the close > 100 day moving average then a buy directive is issued. Once the trade is put on a protective stop of $65 is put into effect and if not hit the algorithm will exit on the close. Notice how the variables that are used in this snippet of code are all inputs. This makes optimizing easy. The computer doesn't need to know the sequencing of the intraday moves with this type of strategy and this is why it is OK to prototype a day-trading algorithm with

daily bars. However, if you get just a few more degrees of complexity in your trading algorithm the results can be highly inaccurate. Take a look at this code of a very simple system.

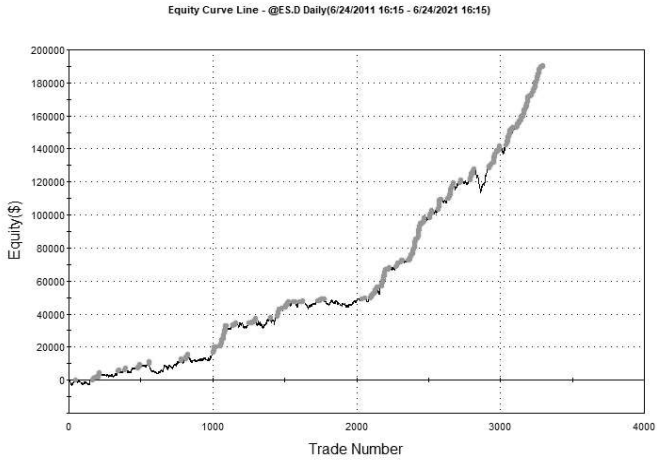
```
inputs:movAvgLen(100),thrustAmt(.25),stopLoss (500), pro-
fitObj (1000);

buy("BThrust") next bar at
      open of next bar + thrustAmt * range stop;

sellShort("SThrust") next bar at
      open of next bar – thrustAmt * range stop;
setStopLoss(stopLoss );
setProfitTarget(profitObj );
setExitOnClose;
```

Video link to Tutorial 8:

https://vimeo.com/580090161/d362ba2e9b

Holy Grail?

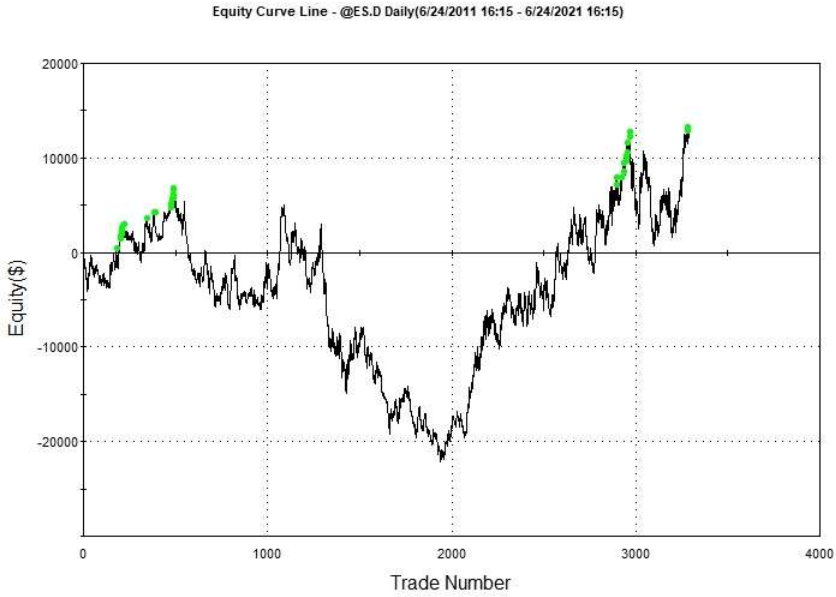
And now the equity curve. Are you ready to bet the farm?



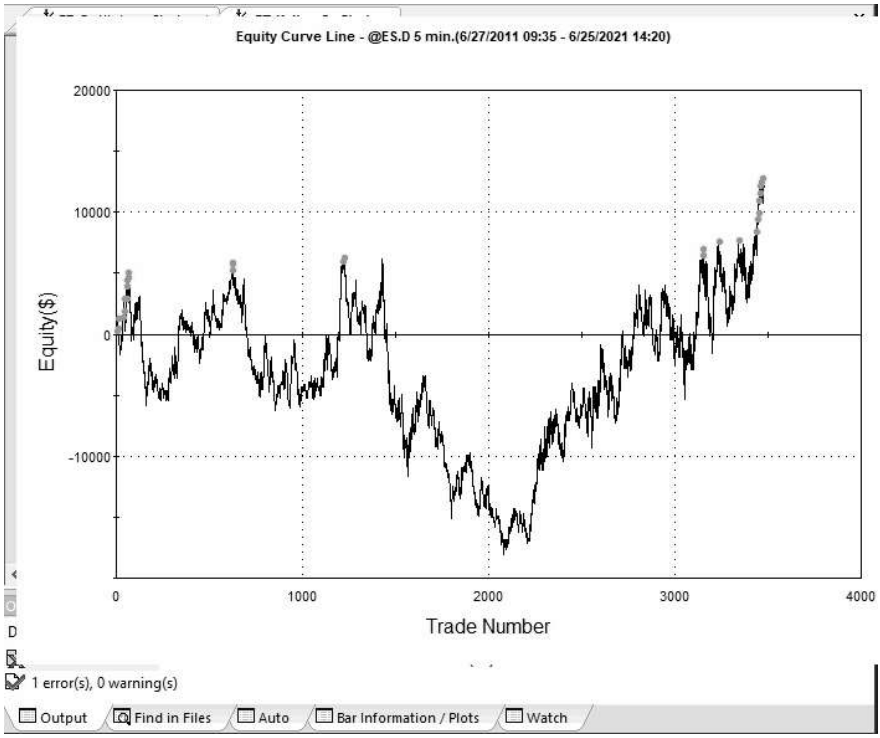
Don't get carried away. When TradeStation needs to determine the sequencing of the daily ebb and flow of the market without the benefit of actually knowing, it can make huge mistakes.

Here we are going to apply the same strategy, but turn **LIB** on with five minute bars. In doing so TradeStation will more accurately determine trade entries and exits. This strategy is highly dependent on the sequence of market highs and lows. Okay I am going to pull the curtain back and reveal the wizard.

The Wizard Behind the Curtain



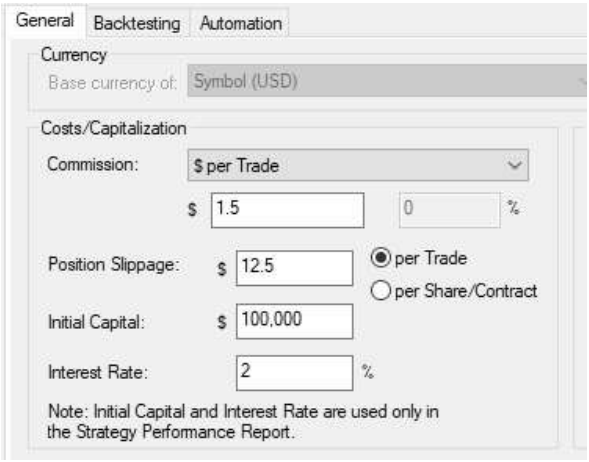
Yes, this is the same exact system. TradeStation uses a formula based on the relationships of the open with the high/low and the close with the high/low and makes an educated guess as to the sequencing of the intraday market movement. If the open is near the low and the close is near the high, then the low of the trading day probably occurred first. The inverse is true about the high of the day occurring first. As you can see this educated guess is not all that accurate. How accurate is the **LIB**, you may ask? Let's test it.



If you review the charts you will see they are quite similar, but not exact. There is a difference between actually using 5 minute bars and the appropriately programmed algorithm and using a daily bar system with **LIB** applied at the same data resolution. But you can prototype a strategy using **LIB**, but be sure to fully evaluate your trading strategy on an intraday basis if you need to. I wished it would be as simple as taking your code for daily bars and simply applying it to intraday data, but it isn't. Once you master the **Foundation Edition of Easing Into EasyLanguage**, you can move up to programming

intraday systems. One last thing before we move on to the next tutorial let's talk about trade slippage or friction. Every trade experiences a commission charge and more likely than not some form of slippage - getting a worse price than you expected. Stop and market orders are notorious for slippage. So it is very important to apply accurate values for commission and slippage. Commission is a fixed value, but slippage can be variable - sometimes you will get your price and sometimes you will not. Market orders can produce zero, negative or positive slippage. The outcome, if you have traded a bunch, is usually negative. Stop orders will provide either zero of negative slippage. When testing and being honest I like to use the current commission rate and 1 tick in and 1 tick out for slippage.

Assume you are trading the mini ES futures - the commission is around $3 and 1 tick is equivalent to $12.50. So in your **Format Strategies:Properties For All** dialog settings do this.



So on entry you will be levied $1.50 for commission and$12.50 for slippage.

The same charges will be levied as well on exit. So a round turn trip will cost $28.00 per trade. Many system vendors would argue that this is worst case scenario and a more accurate slippage value should be used - one that is much less. But remember their intent. Use what you are comfortable with - you can leave it as zero and then simply examine the average trade value. If the average trade is low, then any execution costs will harm performance. If your testing yields a low average trade, then do not get discouraged, its better to know this now then when actual money is on the line. Stress testing an algorithm will pay off big time in real time trading. If you are always entering and exiting on limit order, then you could get away with using zero slippage. But ALWAYS remember to check the option box where the price must trade through the limit price. In real time trading, as I have state previously I have experienced very little success of orders being filled at my limit price when it is touched. This limit order experience is from the Futures universe, stocks might be different because of higher volume, but it is better to be safe than sorry.

What Did Tutorial 8 Teach Us?

- Some day trade systems can be tested using daily bars without the use of **LIB** (look inside bar).

- If a system does not need to know what occurs first, the high or the low of a market move, then you can test with daily bars. A system that enters on the open (long or short) and has either a protective stop or a profit objective (one but not both) then you can test on daily bars.
- A strategy that buys/shorts on an offset off the open (open range break out - ORBO) and exits with a profit or end of day can be tested as well. Only one trade per day can be accurately tested
- You can test more sophisticated day trading algorithms with the use of **LIB**. In this tutorial an ORBO (long and short) with associated profit objective and protective stops was tested with similar results to the same algorithm customized for and used on five minute bars.

TUTORIAL 9: USING INDICATORS TO FILTER TRADES IN YOU TRADING ALGORITHM

Indicators are an attempt to reduce technical analysis down to a generalized mathematical formula. The first book I read on indicators was by Wilder, J. Welles. New Concepts in Technical Trading Systems. Trend Research, 1978. This book, predated most home computers and the calculations were completed either by hand or calculator. To cut down on repetitive calculations, Welles used an exponential moving average like smoothing method for his calculations. Many of the ideas introduced in that book are still prevalent today and is included in most charting packages. In this tutorial we will look at his ADX and RSI indicators to help deter- mine trend and over bought/sold conditions.

Donchian and **Keltner** Channels, an **Bollinger** Bands were used in the first few tutorials to determine entry and exit levels. So the use of indicators will not be new,

but using them to assist other indicators in the trade decision process is new to this book. Also, the concept of a multiple output indicator/function using **Stochastics** will be introduced in this tutorial.

ADX Algorithm

The first algorithm we will program incorporates **ADX** to deter mine if a market is in "trend" mode and if so initiate trades. If the ADX determines that the mode is not trendng, then we will use a different exit mechamism. Before we start programming here is the definition of ADX or Average Directional Index.

From Investopedia.com - ADX is used to quantify trend strength. ADX calculations are based on a moving average of price range expansion over a given period of time. The default setting is 14 bars, although other time periods can be used. ADX can be used on any trading vehicle such as stocks, mutual funds, exchange-traded funds and futures.

ADX - Meaning

- 0-25: Absent or Weak Trend
- 25-50: Strong Trend
- 50-75: Very Strong Trend
- 75-100: Extremely Strong Trend

Basically if the market is moving in one direction without much deviation then you get a high reading. Most markets are considered trending with an ADX above 25 - it is very rare to see a reading much above 50. The ADX indicator is simply a graphical plot of the ADX function. This function is very easy to interface because it just requires one input - length. Many traders like to use 20 bars as the input because that equates to almost a month of data. Remember, since this indicator is lagging (any moving aver- age approach has this property) you are simply hoping the recent market movement will continue for a while longer.

In this tutorial, the Keltner algorithm that was discussed in Tutorial 4 will incorporate the ADX function. Here are the simple rules.

If ADX > 25 then buy/sell short on penetrations of a Keltner Channel of length 20 and 2 X ATR. Exit any position whenever the market moves back through the 20 day moving average. If in a trade and the ADX drops below 20 then exit on the penetration of a 10 day moving average. For right now we will comment out the **SetStopLoss** and **SetProfitTarget**. Try to program this yourself by creating your own EZ_Keltner2 and include ADX(20) in your buy/short order directives and then create two if-then constructs that use a 10 day moving average as bench-

marks to exit if the ADX(20) falls below 20. Did your code look like this?

```
//EZ_WilderADX
inputs: keltnerLen(20), keltnerNumAtr(1.5), profi-
tAmt(1000), stopLossAmt(500);

if c crosses above
      keltnerChannel(close,keltnerLen,keltnerNumAtr) and
      ADX(20) > 25 then
            buy("KeltBuy") next bar at open;

if c crosses below
      keltnerChannel(close,keltnerLen, keltnerNumAtr) and
      ADX(20) < 25 then
            sellshort("KeltShort") next bar at open;

if marketPosition 1 then
begin
      If c crosses below average(c,keltnerLen) then
            sell("LongLiq") this bar on close;
      If ADX(20) < 20 and c crosses below average(c,10)
then        sell("LongLiqADX 20") this bar on close;
end;

if marketPosition 1 then
begin
      If c crosses above average(c,keltnerLen) then
            buyToCover("ShortLiq") this bar on close;
      If ADX(20) < 20 and c crosses above average(c,10)
then        sell("ShortLiqADX 20") this bar on close;
end;
```

```
//setStopLoss(stopLossAmt);
//setProfitTarget(profitAmt);
```

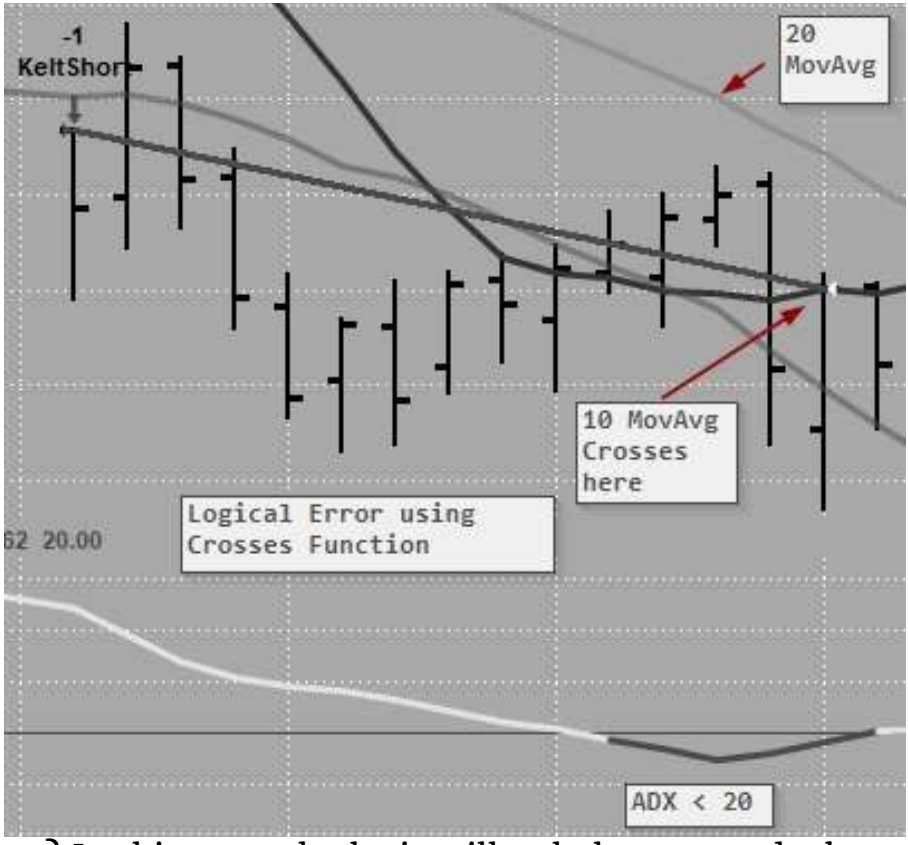
Here is the video link for Tutorial 9:

https://vimeo.com/580747251/c45cc7730f

Code Explanation

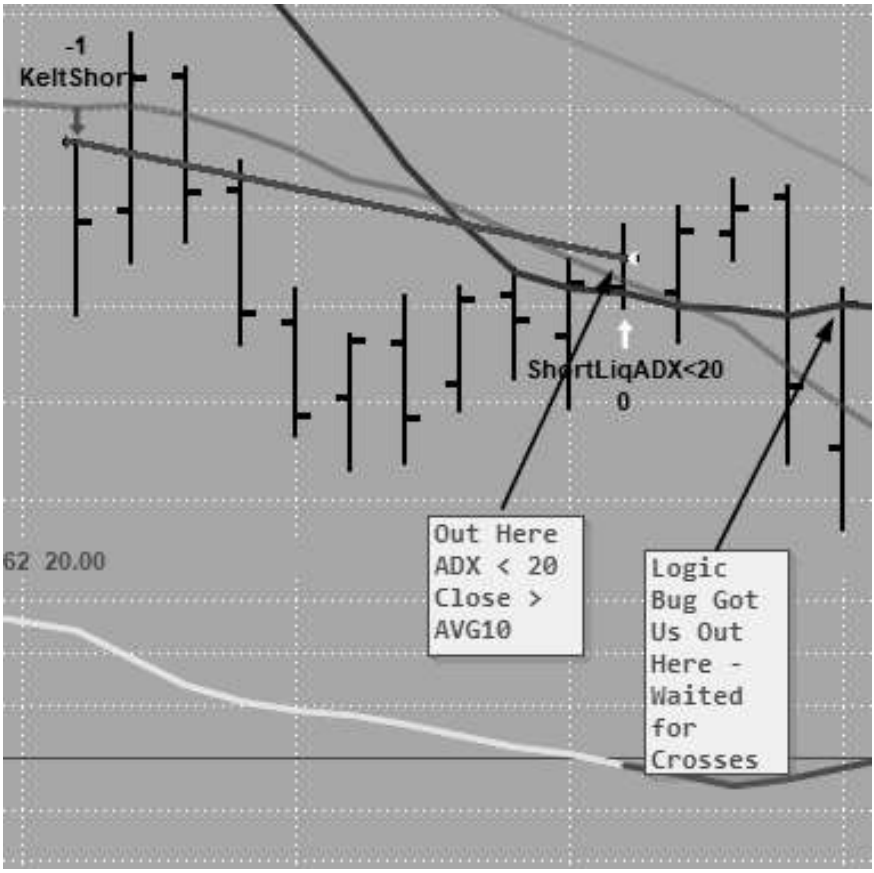
This is how I would personally and initially program the amendments. The if-then that controls the **buy** directive has an additional condition: ADX(20) > 25. The **sellShort** directive has it as well. That was pretty easy. Now notice how the ADX(20) < 20 and c crosses below average(c,10) block was added to the long exit condition. A similar block was added to the short exit as well. In programming you might have arrived at the same location but followed a different route and that is okay - programming style is as unique as the person doing the programming. I used the Keltner strategy as the base for a two reasons: 1) to show how to simply add an indicator as a filter and 2) to present a potential logic bug in the code. There are two types of bugs that are universal to all programming languages and those are: **syntax** and **logic**. The use of the 10-day average was to accelerate an exit when the market entered into a range bound mode - logical, right? However, what happens if the market has already

crossed the 10-day average when the ADX(20) falls below



20? In this case, the logic will only be executed when all criteria are met.

Remember **Crosses** means literally to cross. So our attempt to exit the market earlier than normal may not occur or be delayed; like it was in the prior chart. This is where the once very handy function Crosses causes us a problem. So how do you fix it? Look at the highlighted code after the graph.




```
//EZ_WilderADXFix
if marketPosition = 1 then
begin
      If c crosses below average(c,keltnerLen) then
            sell("LongLiq") this bar on close;
      If ADX(20) < 20 and c < average(c,10) then
            sell("LongLiqADX 20") this bar on close;
end;
if marketPosition = 1 then
```

```
begin
      If c crosses above average(c,keltnerLen) then
            buyToCover("ShortLiq") this bar on close;
      If ADX(20) < 20 and c > average(c,10) then
            buyToCover("ShortLiqADX 20") this bar on
close;
end;
```

It is a simple fix, we just need to eliminate Crosses and use the greater or less than signs. This was indeed a logical bug, but does it make the system better or worse? You will need to test it for yourself.

RSI And ADX Algorithm

The **RSI** (Relative Strength Index) is another creation of Wilder. Here is a definition via Investopedia.com

The relative strength index (RSI) is a momentum indicator used in technical analysis that measures the magnitude of recent price changes to evaluate overbought or oversold conditions in the price of a stock or other asset. The RSI is displayed as an oscillator (a line graph that moves between two extremes) and can have a reading from 0 to 100.

The RSI basically compares the magnitude of up closes to the magnitude of down closes over a specified time period. When a market moves up, the magnitude of the change in up closes is usually greater than the magnitude of down closes so the oscillator will have a higher reading. Here is the base formula.

$$RSI = 100 - (100/(1+RS))$$

RS = Average of 14 day's closes UP / Average of 14 day's closes DOWN

So, if you had 14 consecutive up closes and no down closes then the RSI reading would be 100. If you had zero up closes and 14 down closes then you would have an RSI reading of zero. Whatever goes up must come down, right? So when the RSI exceeds a certain high level, then the market is considered to be overbought. When this occurs a turning point to the downside is predicted. When the RSI falls below a certain low level then a turning point to the upside is expected.

Algorithm Description

Go long when the RSI(14) crosses above 35 and the ADX (14) is less than 20. Cover (sell) when RSI(14) continues to rise and crosses

50. Go short when the RSI(14) crosses below 65 and the ADX(14) is less than 20. Buy to cover when RSI(14) continues to decline and crosses 50.

Code And Explanation

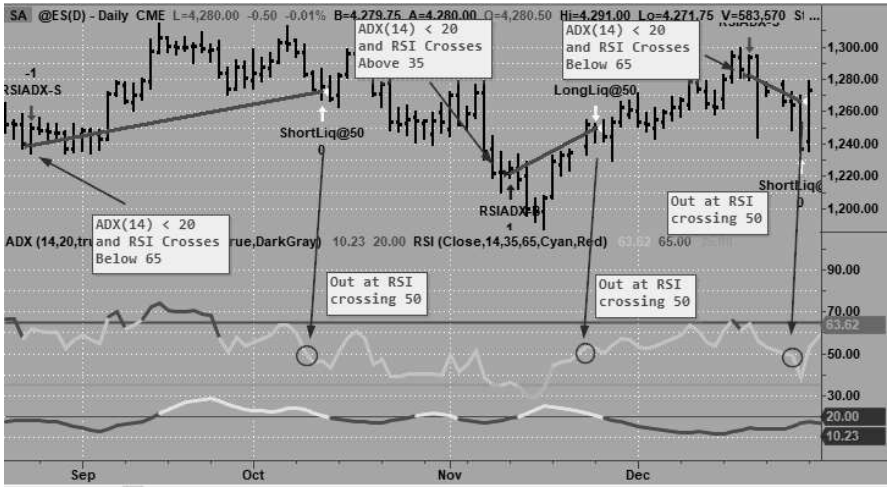
```
//EZ_RSIwADX
```

```
inputs: ADXLen(20), RSILen(20), RSIOBVal(65),
RSIOSVal(35), RSIProfVal(50);



vars: rsiVal(0), adxVal(0), mp(0);

rsiVal = RSI(c,RSILen);
adxVal = ADX(ADXLen);
mp = marketPosition;

If adxVal > 20 and rsiVal crosses above RSIOSVal then
      Buy("RSIADX B") next bar at open;

If adxVal > 20 and rsiVal crosses below RSIOBVal then
      sellShort("RSIADX S") next bar at open;

If mp = 1 then
Begin
      If rsiVal crosses above RSIProfVal then
            sell("LongLiq 50") next bar at open;
end;
If mp = -1 then Begin
      If rsiVal crosses below RSIProfVal then
            buyToCover("ShortLiq 50") next bar at open;
end;
```

This is probably getting very familiar to you, but there are some nuances that need to be explained. Notice I use to variables, **rsiVal** and **adxVal**, to store their respective values. I didn't want to have to keep typing the function

call over and over again in my trade directives so I simply call the functions once and store the values. This should improve efficiency as well. Also, remember in one of the first tutorials when we discussed Bar Arrays? On your first pass of programing an algorithm you might need to know a prior ADX or RSI value, so you might as well us a user defined variable, just in case.

Notice the RSI function requires a price (Open, High, Low, or Close) as the first parameter and the length used in the calculation. Remember a change in high prices might be as important as the change in closing prices, so the RSI function is open to any price change.



If the RSI crosses from below the OverSold value and the ADX < 20 then a buy order is initiated. The thought

behind this is the market has exited the OverSold level and is going to reach across to the OverBought level and since the ADX is less than 20, then the market is theoretically range bound. If the RSI crosses below the OverBought level and the market is range bound as defined by the ADX. then a short position is initiated. Since a short time horizon of 14 bars are used in the calculations, this algorithm is looking to take a quick profit. If the RSI continues to move up after a long position is established, the position is liquidated once the RSI crosses above 50 - the midpoint. The same concept is used for a short trade. Short at RSI level of 65 and cover when the market cross 50 from above. That's it - fairly simple. And I made sure I wanted to us the Cross function!

Using A Multiple Output Function - Stochastics

This portion of this tutorial is a step up in the understanding of programming languages. Just stick with this and you will learn the concepts of functions, procedures, passing by value and passing by reference. In many languages you have two sub program paradigms: function and procedure. A function is a small sub program that returns just one value; functions like the RSI and ADX. Some sub programs are more complicated and need to return multiple values and they are usually called procedures. EasyLanguage combines the two and simply calls

them either a single output function or a multiple output function. The **stochastic** indicator falls in the latter category. To fully engage with the stochastic indicator you need to know multiple components or results of the calculations.

From Investopedia.com again here is the explanation of the stochastic indicator.

A stochastic oscillator is a momentum indicator comparing a particular closing price of a security to a range of its prices over a certain period of time. The sensitivity of the oscillator to market movements is reducible by adjusting that time period or by taking a moving average of the result. It is used to generate overbought and oversold trading signals, utilizing a 0–100 bounded range of values.

Now there are three versions of stochastics. From school.stockcharts.com.

There are three versions of the Stochastic Oscillator available on SharpCharts. The Fast Stochastic Oscillator is based on George Lane's original formulas for %K and %D. In this fast version of the oscillator, %K can appear rather choppy. %D is the 3-day SMA of %K. In fact, Lane used %D to generate buy or sell signals based on bullish and bearish divergences. Lane asserts that a %D divergence is the "only signal which will cause you to buy or sell." Because %D in the Fast Stochastic Oscillator

is used for signals, the Slow Stochastic Oscillator was introduced to reflect this emphasis. The Slow Stochastic Oscillator smooths %K with a 3-day SMA, which is exactly what %D is in the Fast Stochastic Oscillator. Notice that %K in the Slow Stochastic Oscil- lator equals %D in the Fast Stochastic Oscillator.

Sounds rather sophisticated does it not? If you are ever in doubt on how to use an indicator function you can search the TradeStation help by typing the function name and then right click on the name and look up the definition. I can never remember how to call this function so I looked it up and extracted the information that I needed.

Stochastic Funtion Example

Outputs the FastK, FastD, SlowK, and SlowD stochastic values to the declared variables over 14 bars.

```
Vars: oFastK(0), oFastD(0), oSlowK(0), oSlowD(0);


Value1 =
Stochastic(H,L,C,14,3,3,1,oFastK,oFastD,oSlowK,oSlowD);
```

Notice how the lat four variable names in this snippet starts with the small letter "o"? This is TradeStation style that informs you that these values are going to be changed inside the function and you can use them inside

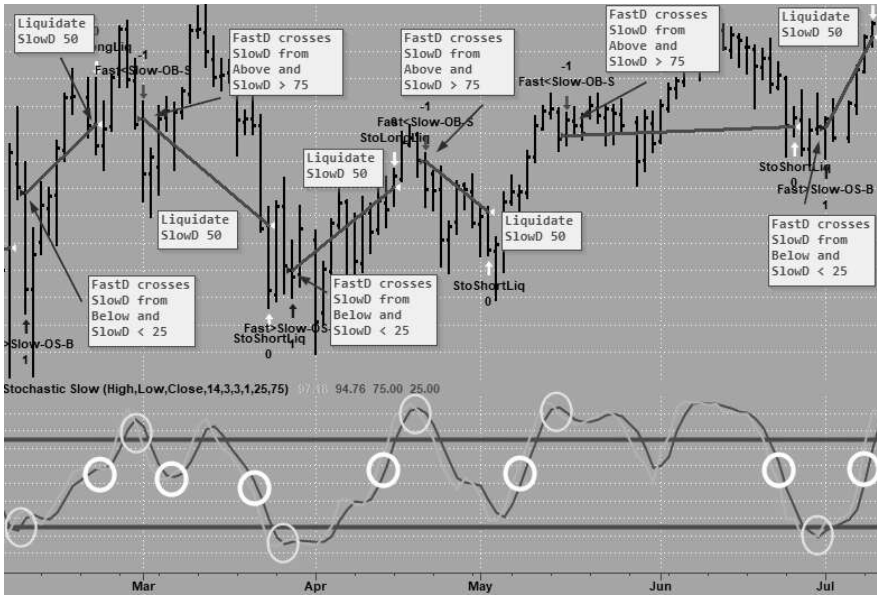
you own code. In my example, I will do just that. Notice how the return value of the stochastic function call is assigned to the built-in variable **value1**. This is known as a dummy variable - it is there just as place holder. You won't use this value in your use of the stochastic function.

Algorithm Description

The stochastic consists of two main values **FastD** and **SlowD**. In addition, like the RSI, there are two regions that indicate an over- bought and oversold market. These value are usually 80 for over- bought and 20 for oversold. In our example we will use 75 and 25 respectively. And we will use the FastD as the trigger line against the SlowD. So if the FastD crosses the SlowD from above and the SlowD > 75 a sell short order will be initiated. On the hand, if the FastD crosse from below the SlowD and the SlowD is < 25 a long order is will issued. Once the SlowD crosses 50 from below, the long position will be liquidated. Once the SlowD crosses the 50 from above the short position will be covered. That is it. We could incorporate the ADX on top of this, but the purpose of this tutorial is to demonstrate the use of a multiple output function.

Code And Explanation

```
//EZ_Stochastic
Inputs:stochLen(14),smooth1(3),smooth2(3),StoOBVal(75),
StoOSVal(25),StoLiqVal(50);
Vars: oFastK(0), oFastD(0), oSlowK(0), oSlowD(0),mp(0);

Value1 =
Stochastic(H,L,C,stochLen,smooth1,smooth2,1,oFastK,oFastD,
oSlowK,oSlowD);

mp = marketPosition;

If oFastD crosses above oSlowD and
      oSlowD < stoOSVal then
            buy("Fast>Slow-OS-B") next bar at open;
If oFastD crosses below oSlowD and
      oSlowD > stoOBVal then
            sellShort("Fast<Slow-OB-S") next bar at open;

If marketPosition = 1 then
Begin
      If oSlowD > StoLiqVal then
            sell("StoLongLiq") next bar at open;
end;
If marketPosition =-1 then
Begin
      If oSlowD < StoLiqVal then
            buyToCover("StoShortLiq") next bar at open;
end;
```



Using these oscillator type indicators provoke similar algorithm design. This code should look very familiar to the RSI and ADX strategy. The most important part of this code is the function call and how the values **oFastD** and **oSlowD** were derived and used in the code. **Value1** was not ever used in the strategy, even though it was used to store the return value of the function.

What Did Tutorial 9 Teach Us?

- How to incorporate indicator functions to help make trading decisions.

- Introduced the **ADX** and **RSI** indicator functions to signal trades when their readings cross overbought and oversold levels.
- How to access multiple output functions and use their individual return values.

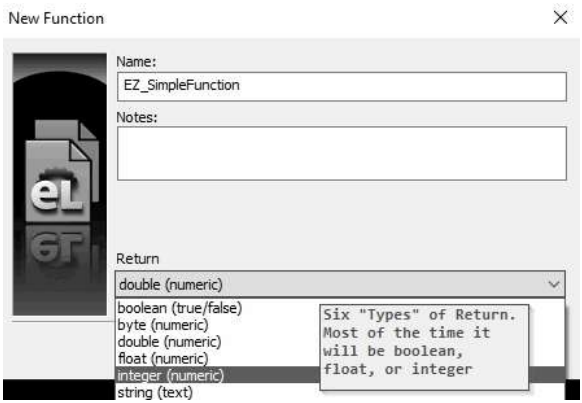
TUTORIAL 10: ALL YOU WANT TO KNOW ABOUT FUNCTIONS

Well Almost.

Simple Function - Returns Just One Value

Here is the formal parameter list (programmer talk) for the RSI function. When you create a function you need to let the computer or TradeStation know what is expected from it and what it (the function) expects from the calling program - technical analysis technique or strategy.

```
inputs: Price( numericseries ), Length( numericsimple ) ;
blah;
Blah;
RSI=50 * (ChgRatio+1);
```



When you create a function from the TDE it will ask you what "Type" of return value will be generated by the

function. You must choose one before it will allow you to proceed it will usually default to **double**. Code for EZ_SimpleFunction follows:

```
//EZ_SimpleFunction
inputs: Price( numericseries ), Length( numericsimple );

vars: highPrice(0), lowPrice(0);

highPrice = highest(price,length);
lowPrice = lowest(price,length);

value1 = price - lowPrice;
EZ_SimpleFunction =
      intPortion((value1 / (highPrice - lowPrice)) * 10);
```

Link to video for Tutorial 10:

https://vimeo.com/581252841/573f62c917

You will notice the code for this simple function is very similar to a strategy except the inputs **Price** and **Length** are not assigned a default numeric value like 20 or 50, a scalar value. In a function, you have to tell TradeStation what "Type" of data that is going to be passed into the input value. In the case of Price, this function expects a **numericSeries** or a list of prices such as **close** or **high**. The **Length** input expects a **numericSimple** which is just

a plain number like 20 or 50. This little function returns the location of the latest price in the range of the prices over *N* bars. So if the return value is 5, then the price is right in the middle of the last *N* bars highest high – lowest low range. Since users passed a wide spectrum of values into functions, default values are not used. Its up to the programmer/trader to provide this information. The return value that is calculated by a simple function must be assigned to the name of the function. Notice the last line of code is the name of the function **EZ_SimpleFunction** = *intPortion((value1 / (highPrice - lowPrice)) * 10);* If you forget to do this, the TDE will let you know.

Multiple Output Function - Returns Multiple Values

This type of function is just like the **stochastic** function. The values it needs to properly calculate **SlowD, SlowK**, **FastD and FastK** is passed into the functions and **oSlowD, oSlowK, oFastD** and **oFastK** are passed back to the user. That is four return values. Here is the function header for the Stochastic.

```
inputs: PriceH( numericseries ),
PriceL( numericseries ),
PriceC( numericseries ),
StochLength( numericsimple ),
```

```
Length1( numericsimple ),
Length2( numericsimple ),
SmoothingType( numericsimple ),

oFastK( numericref ),
oFastD( numericref ),
oSlowK( numericref ),
oSlowD( numericref ) ;
```

Notice how the return values are "Typed" as **numericref**? When you pass a value into a function as a **numericref** you are actually passing the physical memory address of the variable. User defined variables are really what us programmers refer to as "pointers" to memory locations in the computer's memory. You can use these "pointers" to change the value held in the computers memory. Just like we have been doing all along.

```
myValue1 = 10;  puts the number 10 into the memory loca-
tion pointed to by myValue1.
```

When you pass a variable value into a function that expects a **numericSimple** a copy of the value is used and if the function makes a change to the copy, then the original doesn't change. By telling the computer that the **address** of the variable is expected then a copy is not used; the actual address is passed into the function. Once you change the value in the variable's address, then a new value is created and when the variable is passed back to

the calling analysis technique, it will contain the changed value. This is a little complicated for the scope of this book, but it is very important to understand if you want to create your own functions.

Why Would I Want To Create My Own Function

Whenever you discover you are retyping code over and over again that is usually the heads up that you need to create a function; code that can be reused over and over again without having to recreate the wheel.

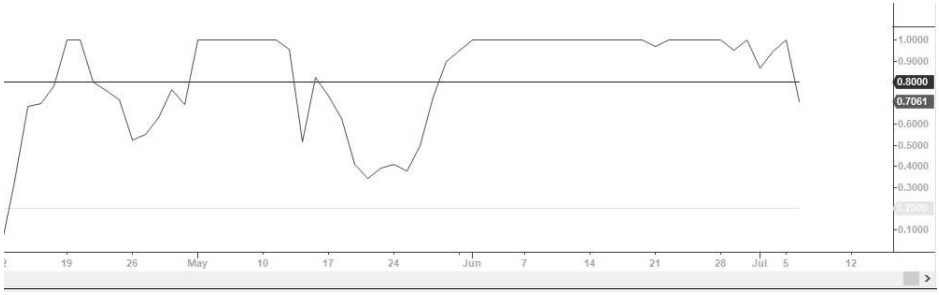
Laguerre RSI Function

The Laguerre version of the RSI is intended to be an improvement on Wilder's version. It is a simple (well sort of) function, that is a great demonstration on how to create a function from scratch. Plus there is some secret sauce in this type of function. From the TDE, goto **New** and select **Function**. The first dialog will ask for the name of the function. Type **EZ_LaguerreRSI** in the **Name** field. The function return type will default to **double** numeric and that is OK. Now type this – just do it, its good practice and its not that much.

```
//EZ_LaguerreRSI function
inputs: myGamma(numericSimple);

vars: L0(0),L1(0),L2(0),L3(0),CU(0),CD(0),L_RSI(0);

L0 = (1 - myGamma) * close + myGamma*L0[1];
```

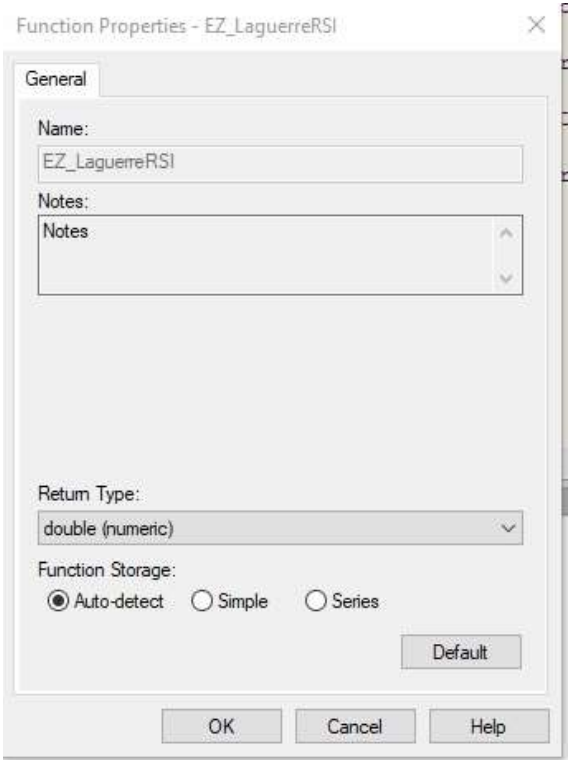
```
L1 = -gamma * L0 + L0[1] + myGamma * L1[1];
L2 = -myGamma * L1 + L1[1] + myGamma * L2[1];
L3 = -myGamma * L2 + L2[1] + gamma * L3[1];

CU = 0;
CD = 0;

If L0 >= L1 then CU = L0 - L1 Else CD = L1 - L0;
If L1 >= L2 then CU = CU + L1 - L2 Else CD = CD + L2-L1;
If L2 >= L3 then CU = CU + L2 - L3 Else CD = CD + L3 - L2;
If CU + CD > 0 then L_RSI = CU / (CU + CD);

EZ_LaguerreRSI = L_RSI;
```

Now verify it. If you have typed everything correctly, you should not get any error messages. If you do just re-check your typing or spelling. You will notice I used the **my** prefix on the variable **Gamma.** Gamma is a reserved word and you should not overwrite reserved/keywords even



if you do you will receive a warning on Verify. In many cases, overwriting a reserved word will no effect on

the execution of your code, but it might! So tread carefully. When I originally used the variable name **Gamma** I received the warning and hence the modification to the variable name. If you like a variable name that is a keyword or reserved word, then simply add **my** on as a prefix (**myGamma**). Now create a chart of a stock or a commodity and insert **EZ_LaguerreRSI** indicator with a **myGamma** of .5 (negative 5). This indicator (we will discuss in the next chapter) calls the EZ_LaguerreRSI function to get the values to plot. Looks rather simple, but there are forces going on behind the scenes that TradeStation or MultiCharts are taking care of for you. With the EZ_LaguerreRSI function, not the indicator, open in your TDE, then right click and select properties at the very bottom of the list of options. You will see the following dialog box. The part that we are interested in here is the **Function Storage**

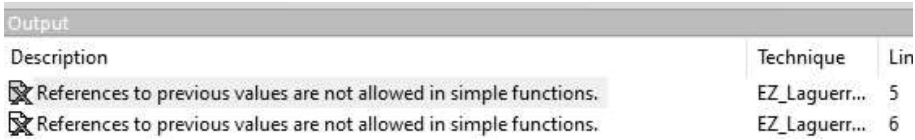


options. Notice you have thee options:

- Auto-detected (default)
- Simple
- Series

Scope – Isn't That A Mouthwash?

Change the **Function Storage** to **Simple** by clicking in that radio button. Once you change the properties of a function you have to re-Verify. So click OK and select Verify under the Build menu. This time you will get an error message that states Reference to previous values are not allowed in simple functions .



You can right click and change back to Auto-Detected and Verify again to get it back to normal. If this makes absolutely no sense to you that is fine. Well at least until I explain what is going on here. In our simple function we use prior bar values of L1, L2 and L3.

```
L1[1] and L2[1] and L3[1] values.
```

Remember when we spoke about Bar Arrays and how they kept track of their prior values or historic values and you accessed them by indexing into them with either a

hard number such as 5 or a variable? This is fine for Analysis Techniques, but functions are a little different story. See the variables declared in a function are usually "local" to just the code in the function This is called the **Scope** of the variable. In most programming languages these variables suffer from amnesia and can't remember from one call to the next. Do you need to worry about the scope of these functions? No not really because the TDE is pretty smart at determining if you need access to prior variable values. But if for some reason the TDE doesn't set the function memory up properly, you will know how to fix this. This auto-sensing is pretty powerful stuff. For your information this is considered a **Series** function. And I will leave it that.

What Did Tutorial 10 Teach Us?

- A function is a subprogram.
- Subprograms usually fall into two distinct categories:
  - Function – one return value
    - Pass variables by value
  - Proecudure – multiple return values
    - Pass variables by reference
- Functions are one evolutionary level shy of object programming

- Function storage is determined by the TDE and can be either Simple or Series – this is based on the need to refer to prior variable values
- **Scope** is a mouthwash, but also applies to programming
- User defined variables might match a keyword and if it does just use **my** or something similar to change the naming convention – it is better to be safe than sorry

TUTORIAL 11: HOW TO PROGRAM AN INDICATOR

Indicators are the distillation of technical analysis. They are attempts to quantify what a technician sees on the chart. Much of the nuance of what the technician sees is not included in an indicator, because it most instances what the eyes see cannot be reproduced in a mathematical equation – not when it comes to supply and demand or market movement or human psychology. Not 100% anyway. Most indicators are used as tools and do help make a trading decision; not because they have captured the essence of the current market, but because they are self-fulfilling. How many people have a 14 period RSI plotted on the same chart at the same time? A bunch I would bet and they react similarly – crowd mentality. You are probably getting an indication, pun intended, that I don't much care for indicators. And you would be partially correct. The shorter term indicators that oscillate may be good for very short term trading like scalping or day trading, but when it comes to developing

trading strategies that can't stand on their own two legs. However, I have seen them work with my own eyes and I have developed many trading algorithms with indicators in concert with price action. So they are important and one needs to know how to program them to solidify a foundation in EasyLanguage. In this short tutorial we will program an oscillator based indicator and one that can be applied onto a price chart.

EZ_LaguerreRSI Indicator

Since we developed the function for this indicator we might as well develop the indicator as well. To the TDE! From the **File** menu select **New** and select **Indicator**. Use the name **EZ_LaguerreRSI** as the name and click OK. Now type the following code.

```
inputs: myGamma(.5);

vars: L_RSI(0);

L_RSI = EZ_LaguerreRSI(myGamma);
Plot1 (L_RSI, "RSI");
Plot2 (.8);
Plot3 (.2);
```

Is that it? Yes, we did most of the programming in the function. Now you can see how the function is an object that we can call over and over again and we don't care what is inside of it. We only care what it can provide us.

The heart of the indicator is the **Plot1** function. You can do a lot of things with this function. Here we simply plotted a line chart that oscillates between 0 and 100. You can change the color and width of the plot by passing those inputs into the function.

```
Plot1 (L_RSI, "RSI",DarkBlue,Default,3)
```

Plot Colors

In this Plot1 function call, DarkBlue and a width of 3 is applied to the plot. Here is a list of colors that can be used in your plots.

EasyLanguage color reserved word	EasyLanguage RGB Value	EasyLanguage legacy color value
Black	0	1
Blue	16711680	2
Cyan	16776960	3
Green	65280	4
Magenta	16711935	5
Red	255	6
Yellow	65535	7
White	16777215	8
DarkBlue	8388608	9
DarkCyan	8421376	10
DarkGreen	32768	11
DarkMagenta	8388736	12
DarkRed	128	13
DarkBrown	32896	14
DarkGray	8421504	15
LightGray	12632256	16

If you want to change the width of your plot line, then you can choose a value from 1 to 6, but you must precede this value with the keyword **Default**. You can leave off the color and width and the plot will default to certain colors that then can be changed by the user via the **Format Analysis Techniques** dialog. The width value is the 5th value in the parameter list that is used by the Plot function. The input or parameter list doesn't have to have all the values such as color or width included but if you do include them, then you must put them in the correct order and the width input is the 5th value so you need to use the **Default** keyword as a place holder in the list.

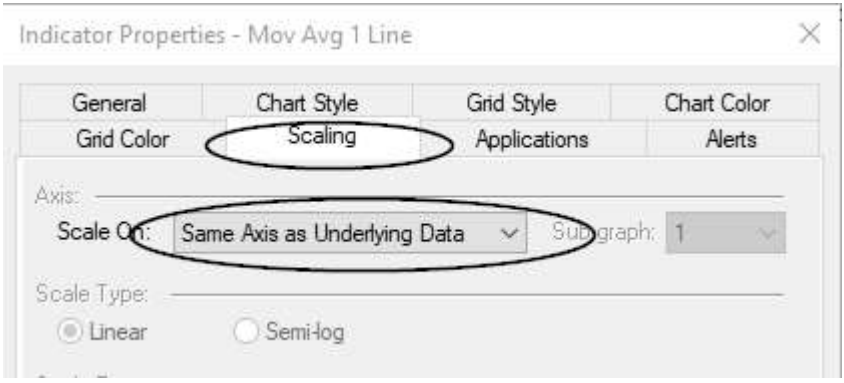
Plot1, Plot2, Plot3 – Each Plot Is Discrete

Another thing you must always use is a different plot function for each plot in your indicator. Notice how the Laguerre RSI is plotted using the **Plot1** function. The boundaries for the **overbought** and **oversold** use **Plot2** and **Plot3** respectively. You can use up to **Plot99.** Ninety nine plots should be sufficient for any indicator, I would think**. Plot2** and **Plot3** only have the plot value passed to them – a constant. So the color is not necessary nor is the width. If you do set them, then the user cannot change them unless they have your indicator code. Setting these value programmatically prevent user customization. Since I like users to have as much control as pos-

sible over the indicators I create, I do not add color or width in my Plot function calls.

Oscillator versus Price Based Indicator

The Laguerre indicator oscillates within a certain boundary and is plotted in a sub-graph. When an indicator doesn't have the same scale as price it will be plotted in a graph with its own scale. An indicator such as a moving average and its associated derivatives, weighted and exponential, should be plotted on the main graph with price since it has the same scale. Also it is important to overlay the actual price to see the interaction of the indicator with price. If you open the **MovAvg1** buit-in indicator from the TDE and right click Properties you will see the following under the Scaling tab.



Subgraph is grayed out so this indicator will automatically be applied to the main price chart.

This is probably a good place to watch a video:
https://vimeo.com/582109515/08cc067abd

Color Based Indicator Value

Many times you will need your indicator to change colors based on the value that is being plotted. In our EZ_LaguerreRSI indicator it would be nice to have the plot change colors if it drops below the oversold boundary and if it rises to the oversold territory. There are different ways to program this but the easiest is to use the **SetPlotColor** function.

```
inputs: myGamma(.5);

vars: L_RSI(0);

L_RSI = EZ_LaguerreRSI(myGamma);

SetPlotColor(1,Cyan);
If L_RSI >= 0.8 then SetPlotColor(1,Red);
If L_RSI <= 0.2 then SetPlotColor(1,Green);

Plot1 (L_RSI, "RSI");
Plot2 (.8);
Plot3 (.2);
```

Notice the conditional branching – the only possibility of a potential color is contained with the universe of Cyan, Red, and Green. You could use different branching with an else statement, but I like top down programming

with as little branching as possible. Here the **Plot1** will be Cyan unless the L_RSI value >= 80 or if its <= 20;

What Did Tutorial 11 Teach Us?

- How to create an indicator from scratch
- How to invoke the Plot function to plot values
- Determine scaling based on class of indicator: price or oscillator
- Change Plot color with the **SetPlotColor** function and conditional branching

TUTORIAL 12: How Do You Debug In The TDE

Learning the syntax of a language is important because without it you can't even take the first step on a long journey. Applying the syntax in a logical manner to solve a problem is just as important. The application of the syntax is where thinking takes over memorization and this is where most new programmers get hung up. It is true that many individuals will never get the concept of creating a complex program and it has nothing to do with intelligence. Organization, understanding logic and sequencing are traits that almost everybody have. Some just have higher levels of these traits. If you are of the trader population, I would think that your analytical tool set would contain a high degree of these traits. If you are a trader and you find it difficult to learn EasyLanguage or programming. then it probably boils down to time spent in front of the TDE and TradeStation. Frustration can set in very quickly when your intentions are not reflected on

the chart. Nearly 100% of my attempts to program an analysis technique, be it an indicator or function or strategy, never works the first time I type the syntax in. Many times the syntax doesn't work right off the bat either, but the computer let's you know where your mistakes are located and provides a helping hand to fix the problem. When the syntax is correct but the logic is wrong, there is nobody but yourself or another's eye to help solve the issue. So your only option to fix a logical error is to **debug**. I love debugging, because unless there is something inherently wrong with the language or the computer, you can fix your code to do what you intend it to do by reviewing, fixing, reviewing and fixing. This cycle might take a few hours, but you will inch a little closer on each iteration.

Modularity

The first thing you can do from the start to prepare for future debugging (do it because you will need to debug) is to break up your logic into different modules. I like to break up my logic in the following modules:

1. Variables: make sure you put these at the top
2. Calculations: do all you calculations in one place only
3. Buy / Long Exit directives: program the logic to get you into or out of a long position first. Once you program this then you can copy past and just

change a portion of the code to invoke a Sell Short / Short Exit directive.

4. SellShort / Short Exit directive: usually similar but opposite the Buy directive.

If you use this top down approach, then you can debug each section separately. The TDE has a debugger, but most of us professional programmers don't use it. Its just so inflexible and difficult to incorporate. The best two tools for debugging EasyLanguage are: the **Print** statement and the EasyLanguage **Print Log**. When you embed the Print statement in your code, all the information is dumped out to the Log window. If you really need to keep the debug information you can also dump this out to a file. Here is some code with an embedded bug.

Print To The EasyLanguage Output Log

You need to see what is going on under the hood once you find out that things are working as they should. Here is an easy to find bug, but let's pretend we don't see it and lets follow the procedure for the this little bug to reveal itself.

```
//EZ_Debug1
//This code has a bug
// can you find it?

inputs: atrLen(30),thrust1Mult(0.75),thrust2Mult(1.5);
```

```
vars:atr(0);

atr = avgTrueRange(30);

If c < c[1] then
Begin
      buy("BDORB-B") next bar at atr * thrust1Mult stop;
      sellShort("BDORB-S") next bar at atr*thrust2Mult
stop;
end
Else
Begin
      buy("SDORB-B") next bar at atr * thrust1Mult stop;
      sellShort("SDORB-S") next bar at atr*thrust2Mult
stop;
End;
```

Create a micro ES.D (MES.D) in TradeStation and apply **EZ_Debug1** strategy. Once you insert this logic you will see that no trades are generated. You have probably already found the problem, but before I show how to definitively root out the problem I am going to explain what the system is trying to accomplish. The concept of a Buy/Short day was introduced by Taylor many years ago. This strategy defines a Buy day as one where today's close is less than yesterday's. Just the opposite defines a Short day. When this occurs the strategy places a stop buy order at the opening of the next day + thrust1Mult * 30 day ATR and a stop short order at the opening – thrust2Mult * 30 day ATR. This is just a simple stop and

reverse strategy that made a ton of money in the 90's. Assuming or pretending that you don't recognize the bug, here is the way to get to the bottom of this tradeless strategy.

Since trades aren't occurring the first thing you probably should investigate is the buy and sell short levels. Since the logic for entering trades is very similar on the buy day (c < c[1]) and the short day (c >= c[1]), then you need to only examine the buy entry and short entry levels for one of the "type" of days. Let's put the debug logic in in C< C[1] section.

```
If c < c[1] then
Begin
      buy("BDORBO-B") next bar at atr * thrust1Mult stop;
      sellShort("BDORB-S") next bar at atr*thrust2Mult
stop;
      print(ELDateToString(Date),
      " B-Day BStop: ",atr * thrust1Mult,
      " B-Day SStop: ",atr * thrust2Mult);
End
```

Now this print statement is very simple to use. All you need to do is enclose what you want to investigate inside parentheses. The first thing we need to see is the date for each bar – I always print this out because it does come in handy sometimes. Speaking of the **Date** keyword there is something you probably need to know. For some odd

reason, TradeStation uses a 7 digit number for the date. So 20210711 (ccyymmdd) format is represented in TradeStation by 1210711. Until you get use to the format, it can be difficult to interpret. Basically when 991231 (yymmdd – pre-millennium change format) advanced to the following day, TradeStation just incremented the year and added a digit – 1000101. I don't know why they just didn't adopt the ccyymmdd format, but they didn't. However, they did provide a function (**ELDateToString**) to provide a visually pleasing format. So that is the first thing that is printed in the print statement. ELDateToString requires the Date keyword. The next thing that is printed is a descriptive string that I came up with, " B-Day Bstop: ". This string describes what is printed next. Notice the two values are separated by a comma. The calculation **atr * thrust1Mult** is then printed out. This time there are no quotation marks because this is not a string but an expression. The print statement recognizes this. Following the buy level calculation is a comma and another descriptive string enclosed in quotes. Finally the short entry level expression is printed out **atr * thrust2Mult**.

```
06/15/2021 B-Day BStop:   30.44 B-Day SStop:   60.89
06/16/2021 B-Day BStop:   30.17 B-Day SStop:   60.34
06/17/2021 B-Day BStop:   30.41 B-Day SStop:   60.82
06/18/2021 B-Day BStop:   30.63 B-Day SStop:   61.26
```

```
06/23/2021 B-Day BStop:   29.44 B-Day SStop:   58.89
07/06/2021 B-Day BStop:   22.49 B-Day SStop:   44.98
07/08/2021 B-Day BStop:   22.91 B-Day SStop:   45.83
07/13/2021 B-Day BStop:   24.16 B-Day SStop:   48.33
```

In 9.5 and 10.0 you access the log through TradeStation and under the **View** menu – **EasyLanguage Print Log**. The current price of the @MES is around 4335.25. So according to our print out we are trying to buy and short and some really strange levels. So we have been able to pinpoint where the problem lies, but what is wrong? Here is the buy directive and level on a Buy day calculation:

```
buy("BD-ORBO-B") next bar at atr * thrust1Mult stop;
```

Do you see it now? I forgot to incorporate the next bar's open price. The current level is not a price level but a measurement of price movement. Let's fix the code to do the following:

```
// The bug has now been fixed.
inputs: atrLen(30),thrust1Mult(0.75),thrust2Mult(1.5);
vars:atr(0);

atr = avgTrueRange(30);

If c < c[1] then
Begin
      buy("BDORBO-B") next bar at
            open of next bar + atr * thrust1Mult stop;
```

```
      sellShort("BDORBO-S") next bar at
            open of next bar - atr * thrust2Mult stop;
//    print(ELDateToString(Date)," B-Day BStop: ",atr *
//    thrust1Mult," B-Day SStop: ",atr * thrust2Mult);
end
Else
Begin
      buy("SD-ORBO-B") next bar at
            open of next bar + atr * thrust2Mult stop;
      sellShort("SD-ORBO-S") next bar at
            open of next bar - atr * thrust1Mult stop;
end;
```

Do you see the bug in this snippet of code?

```
// Another simple bug.

inputs: atrLen(30),thrust1Mult(0.75),thrust2Mult(1.5);
vars:atr(0),mp(0);

atr = avgTrueRange(30);
mp = marketPosition;


If c < c[1] then
Begin
      buy("BD-ORBO-B") next bar at
            open of next bar + atr * thrust1Mult stop;
      sellShort("BD-ORBO-S") next bar at
            open of next bar - atr * thrust2Mult stop;
end
Else
Begin
      buy("SD-ORBO-B") next bar at
```

```
            open of next bar + atr * thrust2Mult stop;
      sellShort("SD-ORBO-S") next bar at
            open of next bar - atr * thrust1Mult stop;
end;

If mp = 1 then
      sell("LongLiq") next bar at entryPrice - 1000 stop;
If mp =-1 then
      buyToCover("ShortLiq") next bar entryPrice +1000
stop;
```

Here we are just setting a $1000 stop loss on long and short entries. If you apply this strategy you will notice its not exiting at all. What's going on? Let's put in some debug code and see what happens.

```
      If mp = 1 then
            print(ELDateToString(Date)," Long Liq",
            entryPrice - 1000);
```

```
06/30/2021 Long Liq 3093.75
07/01/2021 Long Liq 3093.75
07/02/2021 Long Liq 3093.75
07/09/2021 Long Liq 3354.00
07/12/2021 Long Liq 3354.00
07/13/2021 Long Liq 3354.00
07/14/2021 Long Liq 3354.00
```

Well that doesn't look too bad. The level is in the ball-park of a price. But wouldn't you agree that it seems way too far off? How many point are in a $1000 move in the MES futures. Well the **BigPointValue** for the MES is $5 so

a $1000 move would be equal to 200 points. So if the current price 4335.25 and you are long 1 contract the stop loss level should be around 4135.25. Wow, that is really off isn't it? Do you see the problem? We forgot to divide the $1000 loss by **bigPointValue** to get the value into points instead of dollars. Try this :

```
// Another simple bug fixed.

inputs: atrLen(30),thrust1Mult(0.75),thrust2Mult(1.5);
vars:atr(0),mp(0);

atr = avgTrueRange(30);
mp = marketPosition;


If c < c[1] then
Begin
      buy("BD-ORBO-B") next bar at
            open of next bar + atr * thrust1Mult stop;
      sellShort("BD-ORBO-S") next bar at
            open of next bar - atr * thrust2Mult stop;
end
Else
Begin
      buy("SD-ORBO-B") next bar
            at open of next bar + atr * thrust2Mult stop;
      sellShort("SD-ORBO-S") next bar at
            open of next bar - atr * thrust1Mult stop;
end;

If mp = 1 then sell("LongLiq") next bar at
      entryPrice - 1000/BigPointValue stop;
```

```
{If mp = 1 then
            print(ELDateToString(Date)," Long Liq ",
            entryPrice -F 1000);}

If mp =-1 then buyToCover("ShortLiq") next bar
      entryPrice + 1000/BigPointValue stop;
```

If you apply this strategy then you will see exits. A $1000 is rather large move for the MES, so try it with a $100 loss and see if you get more exits. Also take a look how I commented out the logic to print to the print log. I used an opening curly bracket ({) and a closing curly bracket (}). In doing so I can comment multiple lines of code out with out having to put "//" at the beginning of each line. This is where the curly bracket comment comes in real handy.

Formatted Printing

Here are the results of printing out the open, high, low and close price of the Euro Currency Futures to the print log:

```
07/02/2021,   1.19,   1.19,   1.18,   1.19
07/06/2021,   1.19,   1.19,   1.18,   1.18
07/07/2021,   1.18,   1.19,   1.18,   1.18
07/08/2021,   1.18,   1.19,   1.18,   1.19
07/09/2021,   1.19,   1.19,   1.18,   1.19
07/12/2021,   1.19,   1.19,   1.19,   1.19
07/13/2021,   1.19,   1.19,   1.18,   1.18
07/14/2021,   1.18,   1.19,   1.18,   1.18
```

Well that really is all that helpful since the last tic in the Euro Currency future was 1.18095. The last 3 significant digits are missing. Here is the code that provided this useless data:

```
// Another print example.

// I need the EC data for anohter project
// In comma delimitted format
// I could do this in an indicator too!

Print(ELDateToStr-
ing(Date),",",open,",",high,",",low,",",c);
```

The print statement did provide the commas that we needed to import into a spreadsheet application (only because we told it to do so). When the print log doesn't provide enough digits in the output, then you need to apply print formatting to the code to get what you need. Take a look a this bit of code:

```
Print(ELDateToString(Date),",",open:0:5,",",high:0:5,",",
      low:0:5,",",close:0:5);

07/02/2021,1.18655,1.18910,1.18235,1.18605
07/06/2021,1.18830,1.19115,1.18205,1.18395
07/07/2021,1.18365,1.18535,1.17970,1.18215
07/08/2021,1.18080,1.18835,1.17985,1.18550
07/09/2021,1.18600,1.18960,1.18400,1.18920
07/12/2021,1.18920,1.18950,1.18510,1.18730
07/13/2021,1.18760,1.18900,1.17860,1.17960
07/14/2021,1.17920,1.18535,1.17855,1.18440
```

That is much better. Here is the template using a format for output.

print(close:N:M);

The value that is used with the format must be a numeric expression, *N* is the minimum number of integers to use, and *M* is the number of decimals to use. If the numeric expression being sent to the Print Log has more integers than what is specified by N, the Print statement uses as many digits as necessary, and the decimal values are rounded to the nearest value. For example, assume **Value1** is equal to 3.14159 and we have written the following statement:

Print(Value1:0:4);

The numeric expression displayed in the Print Log would be 3.1416. I always use a zero for the ***N*** parameter because it really doesn't make that much of a difference. The important value is what is used for the ***M*** value. If you use a *M* that is less than the number of decimals in a value, then the Print will round the value to the number of decimals that is specified.

Print To A File

Many times I need the data from TradeStation to test on other platforms such as my own TradingSimula-18 or

Excel. A great way to archive this data is to print it out to a file and once its there it is permanent or it is until you delete or overwrite the file.

The following statement sends the information to an ASCII file called mydata.txt:

```
Print(File("c:\data\mydata.txt"), Date, Time, Close);
```

File must include the full path and file name enclosed in quotes. It cannot be a variable or input(notice the backslashes that defines the different directories of where the actual file is contained). Well that is true for the Print statement, but there are ways around that and I will be writing about that in an upcoming book in this series.

What Did Tutorial 12 Teach Us?

- Programming in modules makes debugging much easier.
- Prepare to debug because you will debug – only guarantee I can give in this entire book.
- Use the Print statement and the EasyLanguage Print Log to examine variables.
- Be smart where you put the print statements – make sure its in the direct flow of the program logic or it won't print out the necessary information.

- Don’t waste your time on the TDE Debugger – maybe later on in advanced studying.
- Print formatting can be very important and ultra simple to use.
- Printing to a file helps archive important debug information and data that might be used by a different platform.
- Use commentary syntax to hide lines of code from the computer – // or {}

Tutorial 13 – Additional Programming Techniques – Working with Patterns

This will be the last tutorial for the ***Foundation Edition***. It will be a catch-all for various programming problems that I have come across over the years that were proffered up by new programmers. I will touch on a new construct called a **loop** as well. This will be a great segue into the rest of the books in this series.

Identifying and Working With Patterns

Narrow Range Pattern

In this section we will discuss Toby Crabel's NR patterns and retaining price levels. Toby Crabel did a tremendous amount of research on range compression and expansion. A narrow range (**NR**) bar or day is usually an indicator of range expansion. In other words, the day following an **NR** will usually be a wider than normal bar.

There are numerous ways to identify this pattern, but I will show a simple approach without any shortcuts or advanced programming constructs. There is a builtin nrBar function, but it incorporates OOEL (object oriented EasyLangauge) and therefore cannot be used by MultiCharts. MultiCharts may have a similar function, but the demonstration on how to program this pattern recognition is a

great exercise. Here is the code – do you know what is going on?

Pattern Recognition Code

```
//EZ_NRStrategy

inputs: nrLookBack(7),useTrueRanges(True);
vars: narrowRange(False);

narrowRange = False;

If useTrueRanges then
Begin
      if TrueRange < Lowest(TrueRange[1],nrLookBack-1)
then
            narrowRange = True;
end
Else
Begin
      If Range < Lowest(Range[1],nrLookBack-1) then
            narrowRange = True;
end;
```

This is the beginning of a strategy where a **NR** bar is being identified. The inputs ask how many bars to look back in time to make sure the current bar is a NR and if **trueRanges** should used in place of simple range. A variable with the name **narrowRange** is set to False in the vars: section. By setting it to False, the compiler knows it's a Boolean value. Branching occurs based on whether you want to use TrueRanges or not. It really doesn't matter that much because the same logic is basically used

to identify the NR bar. If the current bar's trueRange (or range) is less than lowest range value going back *nrLook-Back-1* bars then a NR has been located.

The question you are probably asking yourself is why did George do this:

```
Lowest(Range[1],nrLookBack-1)
```

Great question? Why did I start looking back starting with yesterday's range (Range[1] = yesterday's range) and why did I subtract 1 from **nrLookBack**. A NR7 implies that the latest day has the smallest range of the last 7 bars -inclusive. Knowing this I compare today's range to the lowest ranges starting yesterday and going back six bars. Incorporating today's bar in the comparison of the last six bars covers seven total bars; the value that was specified. This looks rather simple, but the counting of bars can be difficult, so how can one make sure this is doing what it is supposed to do before moving on to the next module in the strategy. Since we haven't programmed any trade directives you can copy this code and create either a **ShowMe** or **PaintBar** study and simply paste the code.

Creating a ShowMe With the Pattern Recognition Code

In doing so you can plot the code's output on a chart and visually verify that the code is working. From the TDE go under **File** and select **New ShowMe** and name it something like **EZ_ShowMe1**. Open **EZ_NRStrategy** and copy the code down to the line that states:

```
//Copy above into a ShowMe study
```

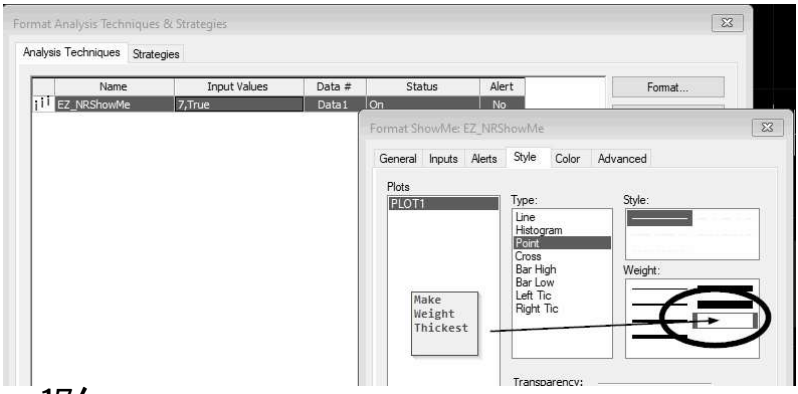
Paste over this existing code in the ShowMe:

```
{ Helpful instructions on the use of EasyLanguage, such as
this, appear below and are contained within French curly
braces {}.  There is no need to erase these instructions
when using EasyLanguage in order for it to function prop-
erly, because this text will be ignored. }

{STEP 1: Replace <CRITERIA> with the logical criteria that
will trigger the placement of a ShowMe marker on the
chart, such as Close > Close[1].
Note that Condition1 is a logical variable, a temporary
holding place for the true-false result of your criteria.
}
```

So your ShowMe should look like this:

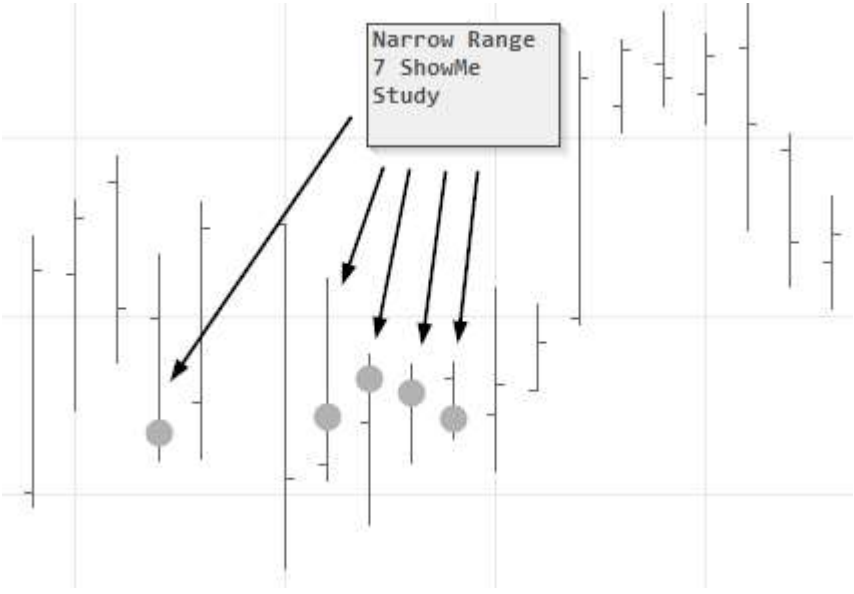
```
//EZ_NRStrategy code for a ShowMe

inputs: nrLookBack(7),useTrueRanges(True);
vars: narrowRange(False);
```

```
narrowRange = False;

If useTrueRanges then
Begin
      if TrueRange < Lowest(TrueRange[1],nrLookBack-1)
then
            narrowRange = True;
end
Else
Begin
      If Range < Lowest(Range[1],nrLookBack-1) then
            narrowRange = True;
end;

Value1 = CLOSE ;
{Leave the following as is.  The plot is not named because
there is only one plot, and the default name Plot1 will be
adequate.  The alert does not include a description be-
cause the alerting criteria and the plotting criteria are
the same, and the description will be redundant. }
if NarrowRange then
begin
      Plot1( Value1 ) ;
      Alert ;
End ;
```



Value1 is set to the closing price

of the bar and this is where the **ShowMe** will plot. Replace **Condition1** with our **NarrowRange** Boolean variable and then Verify. Once you get it verified – might take a couple of tries, apply into a @Cl futures chart. You should see some small dots at the closing prices of NR7 bars. You might want to enlarge the dots to circles. Right click the chart and Format the Analysis Technique and select the heaviest weight. If all goes well then you should see something like this on your @CL chart.



Now see if your logic is correct. Click on the range of a bar with the big red dot and see if it is indeed the smallest range of the last 7 bars. It should be. Now that was neat way to validate our programming logic.

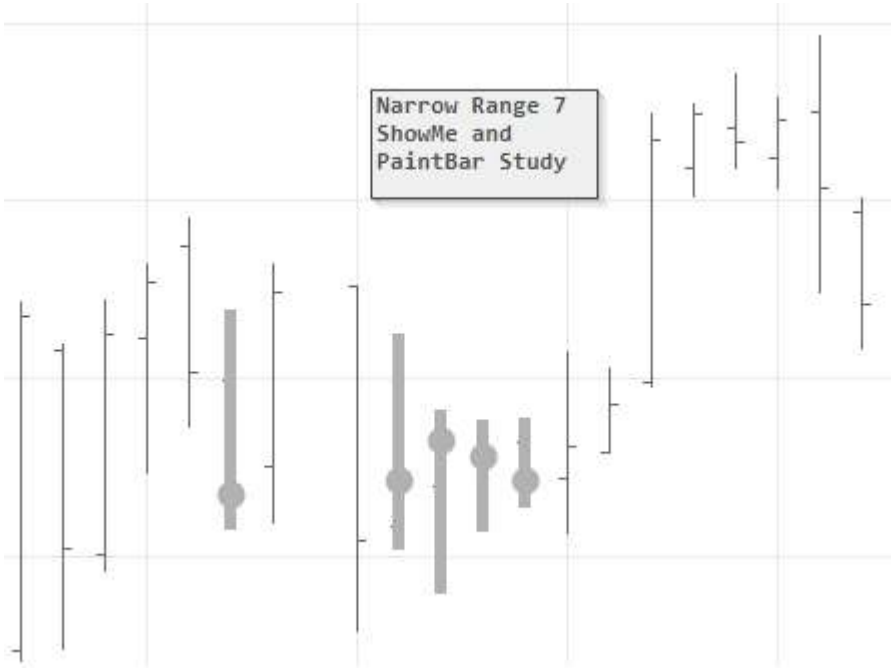
Use The Same Logic To Create a PaintBar Study

Let's do the same with a **PaintBarStudy**. From the TDE go under File and create a **New PaintBar**. Give it a name like **EZ_PaintBar1**, but this time select **Simple** from the drop down Template box. Now make your code look like this:

```
//EZ_NRStrategy for PaintBar

inputs: nrLookBack(7),useTrueRanges(True);
vars: narrowRange(False);

narrowRange = False;

If useTrueRanges then
Begin
      if TrueRange < Lowest(TrueRange[1],nrLookBack-1)
then
            narrowRange = True;
end
Else
Begin
      If Range < Lowest(Range[1],nrLookBack-1) then
            narrowRange = True;
end;

//Copy above into a ShowMe study and PaintBar Study
Value1 = HIGH ;
Value2 = LOW ;
```

```
{ Leave the following as is.  The plot is not named be-
cause there is only one PaintBar plot - with two sub-plots
- and the default names Plot1, Plot2 will be adequate.
The alert does not include a description because the
alerting criteria and the plotting criteria are the same,
and the description will be redundant. }

if narrowRange then
begin
      PlotPaintBar( Value1, Value2 ) ;
      Alert ;
end ;
```

The **PaintBar** will color the entire bar from high to low. Check this out here and on your screen.



Basically this study replicates the ShowMe, but it does show how to create a PaintBar study. This paints the entire bar from high to low, but you can change the portion of the bar that you want painted by changing **Value1** and **Value2**.

Narrow Range Pattern Strategy

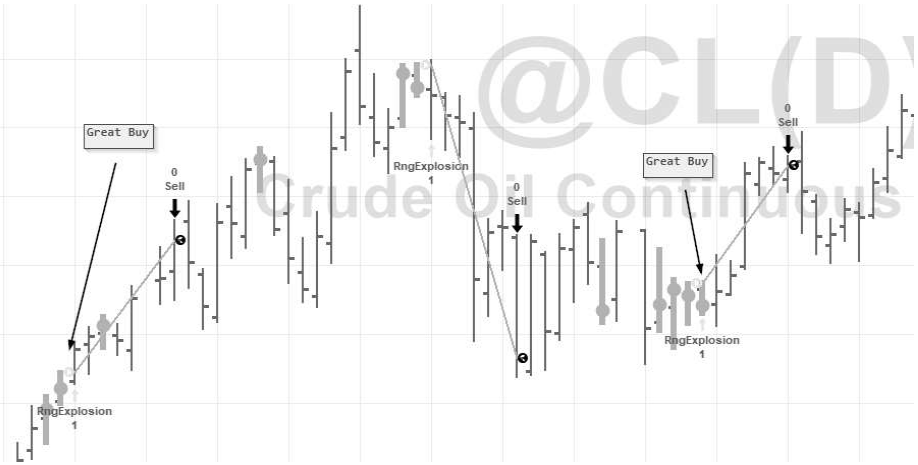
We now know that the pattern recognition code is working so now how can we use it to create a trading system. If one NR7 day indicates the potential for range expansion what would two in a row indicate. I don't know but we can test it. Let's start with the code:

```
//EZ_NRStrategy

inputs: nrLookBack(7),useTrueRanges(True),numBarsInT-
rade(5);
vars: narrowRange(False);

narrowRange = False;

If useTrueRanges then
Begin
      if TrueRange < Lowest(TrueRange[1],nrLookBack-1)
then
            narrowRange = True;
end
Else
Begin
      If Range < Lowest(Range[1],nrLookBack-1) then
            narrowRange = True;
```

```
end;

//Copy above into a ShowMe study and PaintBar Study

if narrowRange and
      narrowRange[1] then
      buy("RngExplosion") next bar at
            h + minMove/priceScale stop;


If barsSinceEntry > numBarsInTrade then
      sell this bar on close;
```

This should be getting very familiar to you by now. We have the input and variables module. Then the calculation module follows. This module also includes branching based on the user's input. Then the buy directive and sell directive finishes it off. Some things may look a little foreign in this code because it probably is the first time you have seen it. Notice how I determined if two days were NR7s in a consecutive basis. I looked at the current NR7 and the prior NR7[1] and then used Boolean Algebra by using the AND to see if both were true. If both were true, then the stop buy order was placed at today's high plus one minimum tick. To get the minimum tick simply divide **minMove** by **priceScale**. In the case of crude oil the **minMove** is 1 and the **priceScale** is 100 so a minimum tick is 0.01. If **barsSinceEntry** (keyword) is greater than

numBarsInTrade (user input -> 7) then an exit occurs. Here is a chart of our work:



If you look at the chart you see the code is doing exactly what we told it to do, and more importantly what we intended.

The Tale of Two Algorithm Paradigms

Over the years, I have discovered there are two different paradigms when describing and or programming a trading system. The first is the easier to program and can be used to describe a system where events occur on a consecutive bar basis; just like the NR pattern strategy The other paradigm can be used to describe a trading system that needs things to happen in a sequence, but not necessarily on a consecutive bar-by-bar basis. Unfortunately, the latter paradigm requires a heightened level of description and is more difficult to program. Many

trading systems can fit inside a cookie-cutter design, but most traders are very creative, and so are their designs. Don't fret, though, because any idea that is reducible to a sequence of steps can be described and programmed. I have coined names for these two paradigms:

- (**VBLS**) the variable bar liberal sequence paradigm
- (**CBES**) the consecutive bar exact sequence paradigm

Multiple Step Strategy or How I Learned to Love the FSM

Programming algorithms that occur on a consecutive bar basis usually, but not always, look back in time. Programming algorithms that progresses through time and doesn't have to occur on a consecutive bar basis usually moves from left to right on the chart. Here is an example of a CBES type algorithm:

The close of a bar must close above the 20-day upper Bollinger band and then a NR7 occurs above the 20-day moving average. Once these conditions are met, then a stop buy is placed at the high of the NR7 plus one minimum tick. This is a very simple strategy to program if I were to introduce the concept of Finite State Automata or Machine, but we aren't there yet and I will hold off on this for another book in the series. Now that doesn't mean we can't program this because we can – just re-

quires a few more variables and if-then constructs (edging close to a Finite Stat Machine). Okay lets attack this problem head on and program the entry technique; the exit technique will be based off a number of days in the trade for simplicity sake. I will present the code for this strategy, but don't be too upset if you don't understand every line – I will go through each one so that by the end you will know exactly how this little strategy was programmed.

```
  //EZ_AlmostFSM

inputs: nrLookBack(7),useTrueRanges(True),BBandLen(20),
            BBandNumStd(2),numBarsInTrade(5);
vars: narrowRange(False),mp(0),upperBand(0),midBand(0);
vars: entryTrigger(0),entryLevel(0);

narrowRange = False;
// we could put the search for a NR in a function
If useTrueRanges then
Begin
      if TrueRange < Lowest(TrueRange[1],nrLookBack-1)
then
            narrowRange = True;
end
Else
Begin
      If Range < Lowest(Range[1],nrLookBack-1) then
            narrowRange = True;
end;
```

```
upperBand = bollingerBand(c,BBandLen,BBandNumStd);
midBand = average(c,BBandLen);

If close > upperBand then
      entryTrigger = 1;

If entryTrigger = 1 then
Begin
      If narrowRange and
      c < upperBand and
      c >  midBand then
      begin
            entryTrigger = 2;
            entryLevel = high + minMove/priceScale;
      end;
end;

If entryTrigger = 2 then
      buy("Trigger=2:B") next bar at entryLevel stop;

mp = marketPosition;

If c <= midBand or mp = 1 then entryTrigger = 0;

if(barsSinceEntry = numBarsInTrade) then sell("TradeExpi-
res") next bar at open;
```

The key variable that keeps this almost machine running is **entryTrigger**. This variable is modified as the market works through the different logical criteria. This is a multiple step algorithm where you have to keep track

of the STATE of **entryTrigger** (this is where the STATE comes from in Finite State Automata.) This one variable causes a signal to be fired or not. Okay let's do a post-mortem on the code.

The inputs and variables module should look familiar to you now. We are going to put anything that we might need for later use into these two statements. When programming, it's alright to adapt these statements to what you need after the fact. Its like when you pack for vacation and you try to remember everything you need but as you know that is almost impossible. When programming a new idea, start out with what you think you might need. While you are programming you will come across the need for either another input or variable name. Just like on vacation you can run to the local grocery store and get what you either forgot or needed. Unless you are hiking the Appalachian Trail as a vacation and then your are simply out of luck. Programming, as you will find, is much easier than hiking.

Since I have already programmed this before writing the description, everything you need is right here. This strategy will require the user to input (for their convenience and optimization capabilities) the number of **nrLookBack** bars to search for a NR bar, to **useTrueRanges** or not, the number of bars and standard deviations in the Bollinger Band calculation, and lastly the number of

bars to hold the trade. The variables that we will need will be the conditional (True/False) **narrowRange**, market position holder **mp**, **entryTrigger**, **entryLevel**, the **upper** and **mid** bands of the Bollinger calculation.

The next module or section of the code searches for an NR7 – this should look very familiar as we just did this. The upper and midBands are then calculated. Now this is where we start the Machine (a not so fancy word for Automata).

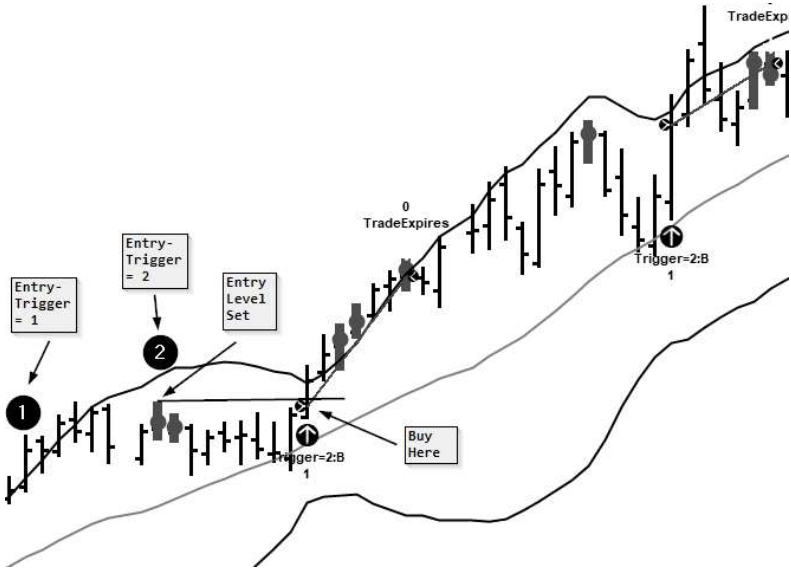
```
If close > upperBand then
      entryTrigger = 1;

If entryTrigger = 1 then
Begin
      If narrowRange and
      c < upperBand and
      c >  midBand then
      begin
            entryTrigger = 2;
            entryLevel = high + minMove/priceScale;
      end;
end;
If entryTrigger = 2 then
      buy("Trigger=2:B") next bar at entryLevel stop;

mp = marketPosition;

If c <= midBand or mp = 1 then entryTrigger = 0;
```

To fire off a trade entry signal **entryTrigger** must be equal to 2 and the price must rise to or above **entryLevel**. How does **entryTrigger** get to a value (State) equal to 2? It must go through logic to move from 0 to 2 – patterns or conditions must be met for the progression to take place. **EntryTrigger** starts out at 0 and then whenever the close is greater than the **upperBand** its value is increased to 1. At this point (when **entryTrigger** is set to 1), the computer starts to look for a NR7 that lies between the upper and mid bands.

```
If narrowRange and
c < upperBand and
c > midBand then
begin
      entryTrigger = 2;
      entryLevel = high + minMove/priceScale;
End;
```

If a **NR7** is found and it indeed lies between the two bands then **entryTrigger** is incremented to 2 and the **entryLevel** is set to the high of NR7 + minimum tick. The machine is now in second gear and possesses an entry level or in other words **entryTrigger** is set to 2 and the **entryLevel** is set to the NR7 high + minTick. You can look at it either way. The strategy is cocked and ready to fire and is only waiting for the market to move above the NR7

high. This could take some time as you can see in the following chart.



The first step is for the market to close above the **upper-Band (1)**. The second step in the process if for a **Nr7 (2)** day to form between the upper and mid bands. Once this occurs the machine is cocked and ready to fire whenever the market exceeds the NR7 high. This occurs a few days later **(Buy Here)**. Once a long position is established **entryTrigger** is reset to 0. **EntryTrigger** is also reset to 0 if the market falls below the midBand. See if the next trade makes sense in the chart. An exit occurs whenever **numBarsInTrade** transpires.

A Narrow Range and Inside Day Pattern

Do you think you can convert our NR logic into a function and also force the NR to be an inside bar? I think you can. First off head back to the TDE and create a function called **EZ_NRandID** and set the return value as Boolean (true/false). You are going to need the same inputs that we used in the NR search logic: number of bars to look back and to use **trueRanges** or not. So with the EZ_NRShowMe opened copy the following logic into your new function.

```
inputs: nrLookBack(7),useTrueRanges(True);
vars: narrowRange(False);

narrowRange = False;

If useTrueRanges then
Begin
      if TrueRange < Lowest(TrueRange[1],nrLookBack-1)
then
            narrowRange = True;
end
Else
Begin
      If Range < Lowest(Range[1],nrLookBack-1) then
            narrowRange = True;
End;
```

Notice we only copied the code that determines the status of an NR bar. One of the core concepts of object

oriented programming is the reuse of code. By reusing this code to create this function you are easing your way into OOP. Now functions cannot accept actual values in the input statement, they need to know what type of value to catch. Basically this is the formal parameter list and it just needs to know what to receive from the calling strategy or analysis technique – think of it as a catcher wanting to know what pitcher is throwing. So let's change the inputs to this:

```
inputs:
nrLookBack(numericSimple),useTrueRanges(TrueFalse);
```

Now what makes an inside bar? An inside bar or day is where today's high is less than yesterday's and today's low is greater than yesterday's low. Simple enough. Now how do we incorporate that into our narrow range logic. Well, we need to know that the bar we are looking at is an NR bar and we have that logic already programmed. Now if the bar is a NR bar then we need to check the high and low of the bar against yesterday's high and low. What do you think about this?

```
EZ_NRandID = False;
If narrowRange and
      high < high[1] and low > low[1] then
            EZ_NRandID = True;
```

Here narrowRange is "Anded" with high < high[1] and low > low[1]. If this Boolean expression is true then we know we have a **NR_ID** bar. If not then the bar does not fit the criteria. Remember functions have to return a value and that is accomplished by giving the name of the function, in this case **EZ_NRandID**, a value and since this function is of a Boolean (true/false) type it expects either a True or False value. Notice how I first assign the function name a False value? This function is binary in the sense it either returns True or False, so I go ahead and assign a False to the value. If the pattern NR_ID is true, then this value is overwritten with True. This is an nuance when it comes to functions. It always returns whatever is the last value that is assigned.

Press the Accelerator to the Floor – Programming the Tom Demark Sequential Paintbar

Whoa! Whoa! Whoa? Wasn't this supposed to be a Beginners Guide and not a slap me upside the head with complicated code and expect it to be understood door stop. Well yes it was supposed to be that and it still is. The purpose of this sidebar is to show how something really complicated can be broken down into rudimentary components and digested in that manner. What if I told you that I can name that tune or program a good portion of that pattern in less than 29 lines of code? Would you be interested then? Let's do it. First off, many of you

may not know what the Tom Demark Sequential pattern is so I will lean on Perry Kaufman (*New Trading Systems and Methods 4th ed., Perry J. Kaufman, p.148*) to help explain this somewhat complicated pattern.

> *Tom DeMark created a strategy called a sequential that finds an overextended price move, one that is likely to change direction and takes a countertrend position.*
>
> *To get a buy signal, the following three steps are applied to daily data:*
>
> ***1) Setup. There must be a decline of at least nine or more consecutive closes that are lower than the corresponding closes four days earlier. If today's close is equal to or greater than the close four days before, the setup must begin again.***
>
> *2) Intersection. To assure prices are declining in an orderly fashion rather than plunging, the high of any day on or after the eighth day of the set up must be greater than the low of any day three or more days earlier.*

3) Countdown. Once setup and intersection have been satisfied we count the number of days in which we close lower than the close two days ago ***(doesn't need to be continuous – a hint to use a FSM)****. When the countdown reaches 13, we get a buy signal unless one of the following occurs:*

a. There is a close that exceeds the highest intraday high that occurred during the setup stage.

b. A sell setup occurs (nine consecutive closes above the corresponding closes four days earlier).

c. Another buy setup occurs before the buy count-down is completed.

Traders should expect that the development of the entire formation take no less than 21 days, but more typically 24-39 days.

Okay – time to get down to brass tacks. Let's jump into the Setup portion of the pattern. This cannot be programmed or can it? Really?

Using For Loops and Checksums

Let's start with checksums; a checksum is a sequence of numbers and letters used to check data for errors. If three conditions need to be true to meet a criteria and you increment a checksum variable every time one of the conditions is true, the variable must be equal to or greater than thee for the criteria to be met. If the checksum is less than three, then you know the criteria wasn't met. That's fairly simple. The if-then and for-loop constructs are probably the two most important programming ideas. Or building blocks, in almost any programming language. In math, a loop is similar to a series represented by sigma notation; there is a begin and an end and a function in between. If the for-loop is so important why wasn't it introduced way before Tutorial 13? You have already seen a for-loop in this book, but it was disguised as a function. The **highest** function is really just a loop going back in time and retaining the highest value over the look back time period.

For Loop in EasyLanguage

Before I show you the universal for-loop template, let's take a look at how the computer can use a loop to find the highest of the last *N* bars:

```
Vars: index(0),highestValue(0),nBars(10);

highestValue = 0;
```

```
for index = 0 to nBars -1
begin
      if(high[index] > highestValue) then
            highestValue = high[index];
End;
```

Notice all **for-loops** must use a **begin** and a corresponding **end**.

Now here is the code that determines the SetUp phase of Sequential. The explanation follows:

```
Vars: index(0),checkSum(0);


checkSum = 0;
for index = 0 to 8
begin
      if(c[index] < c[index+4]) then checkSum = check-
Sum+1;
end;
```

First off we use our old friend checkSum to keep track of things for us; initially it is set to 0. Remember when we spoke about **Bar Arrays** and how all declared variables and most built in data were really lists that could be indexed historically? You know close[1] is yesterday and close[2] is the day before yesterday. Today's close can be represented by either close or close[0]. The value in the index can be a variable too not just a hard coded number. So in the small snippet of code, above or on the prior

page, uses an index (literally index) to look into the close price array or list. The **for-loop** starts at 0 and ends at 8; nine total iterations. Remember if you start counting at 0 instead of 1 you will have an extra element or number in your list. If **index** starts at 0, then the first conditional statement reads:

```
Index is 0
If close[0] < close[0+4] then checkSum = checkSum + 1;
```

This compares today's close to the close 4 days prior. In the next loop **index** increments to 1, so:

```
Index then increments to 1
If close[1] < close[1+4] then checkSum = checkSum + 1;
```

This compares yesterdays close to the close 4 days prior to yesterday or close[1] < close[5]. This loop continues until **index** finally takes on the value 8:

```
Index is 8
If close[8] < close[8+4] then checkSum = checkSum + 1;
```

This translates in to close[8] < close[12]. Now if each conditional test in each iteration of the loop is True, then checkSum should be equal to 9. If checkSum is less than nine, then one of the comparisons failed. The variable following the keyword **for** takes on the values inside the

range from the start to end points inclusive. What do you think this does:

```
For index = 10 downTo 0
```

Did you guess it? This decrements **index** from 10 to 0 for a total of 11 loops. Here is the SetUp pattern code in its entirety:

```
[LegacyColorValue = true];

{EZ_Sequential Setup
 Paint the Sequential Setup Pattern}
Vars: index(0),checkSum(0);

checkSum = 0;
for index = 0 to 8
begin
      if(close[index] < close[index+4]) then
            checkSum = checkSum + 1;
end;
if(checkSum = 9)then
begin
      for index = 0 to 8
      begin
            PlotPaintBar[index](High[index],Low[index],
                  "Sequential",YELLOW);
      end;
end;
checkSum = 0;
for index = 0 to 8
begin
      if(close[index] > close[index+4]) then
```

```
            checkSum = checkSum + 1;
end;
if(checkSum = 9)then
begin
      for index = 0 to 8
      begin
            PlotPaintBar[index](High[index],Low[index],
                  "Sequential",RED);
      end;
end;
```

Twenty nine lines if you don't count the comments. Since we have basically covered the checkSum, we can skip that explanation. The interesting twist to this code is how price bars can be retroactively painted. If checkSum for the buy SetUp is equal to 9, then the code to paint the bars is executed. This code is also included in a loop. Take a look at this statement:

```
PlotPaintBar[index](High[index],Low[index],"Seq…",RED)
```

Notice how **index** is not only used in the price lists, but as an offset for the PlotPaintBar function.

```
PlotPaintBar[0] - paints today
PlotPaintBar[1] - paints yesterday
PlotPaintBar[2] - paints the day prior to yesterday
```

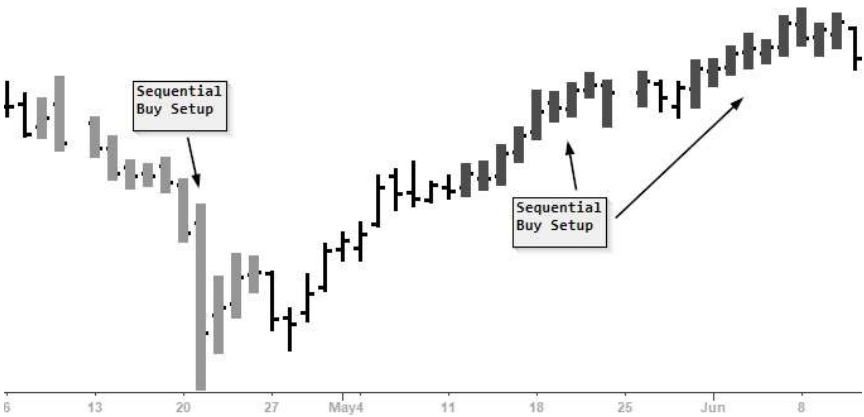
Using the same index for the bar to paint and its respective high and low keeps the PaintBar in synch. If it is

out of synch then a vertical line from a high and low will be drawn on the chart somewhere. If we were to say,

```
PlotPaintBar[1]((High[2],Low[2],"Seq…",RED)
```

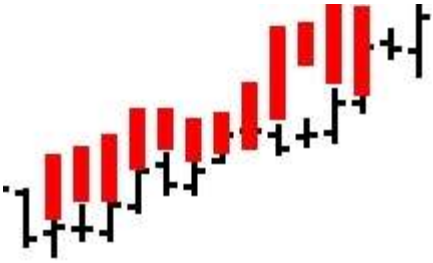
Then a line would be drawn at the horizontal placement of bar[1] but use the high[2] and low[2] range to paint. The bar and the plot would be out of synch.

Here is the plot of the Sequential SetUp Paintbar as we have programmed it, with everything in synch.



Now let's do this:

```
PlotPaintBar[index+3](High[index],Low[index],"Seq…",RED);
```



Bar 3 is painted with the high and low of Bar 1. So you can see how important it is to synch the PaintBar and ShowMe analysis techniques.

Do You Want to See the Complete Sequential Buy SetUp?

Just for the fun of it, let's take a look at the complete Sequential Buy Setup Paintbar. See if you can make heads or tails out of the logic. This builds upon the **for-loops** and checksums that were just mentioned.

```
Vars: index(0),checkSum(0),
      state(0),state1BarNum(0),
      state1High(0),state2BarNum(0),longCountDownCnt(0);

{Buy SetUp - the setup follows the
 Consecutive bar exact sequence paradigm or CBES followed
 by Variable bar liberal sequence paradigm
 using a State Machine to complete the pattern}

checkSum = 0;
for index = 0 to 8
begin
      if(close[index] < close[index+4]) then
            checkSum = checkSum + 1;
end;

If checkSum = 9 and state = 0 then
Begin
      state = 1;
      state1BarNum = barNumber;
      state1High = highest(h,9);
      for index = 0 to 8
      begin
```

```
            PlotPaintBar[index](High[index],Low[index],
                  "Seq.SetUp",red);
      end;
end;

If state = 1 then // found the setup
Begin
      if high[1] > low[3] or high > low[2] then
      begin
            state = 2;
            state2BarNum = barNumber;
            If state2BarNum = state1BarNum then
                  PlotPaintBar(High,Low,"Seq.SetUp",blue)
            Else
            begin
                  for index = 0 to barNumber -
                              maxList(1,(state1BarNum +
1))
                  begin
                        PlotPaintBar[index](High[index],
                                    Low[index],
                                    "Seq.SetUp",blue);
                  end;
            end;
      end;
end;
If  state = 2 then // intersection found
begin
      if c < l[2] then longCountDownCnt=longCount-
DownCnt+1;
      checkSum = 0;
      for index = 0 to 8
      begin
            if(close[index] > close[index+4]) then
```

```
                checkSum = checkSum + 1;
      end;
      If checkSum = 9 or h > state1High then
      Begin
            state = 0;
            longCountDownCnt = 0;
            PlotPaintBar(High,Low,"Seq.SetUp",cyan);
      end;
      If longCountDownCnt = 13 then state = 3;
end;

If  state = 3 then //completed the pattern
begin
      for index = 0 to barNumber - (state2BarNum+1)
      begin

            PlotPaintBar[index](High[index],
                  Low[index],"Seq.SetUp",yellow);
      end;
      state = 0;
      longCountDownCnt = 0;
end;
```

You can tell this is a state machine because of the word **state.** Why use this algorithm paradigm for complicated patterns? If you don't, then you need to keep track of a ton of different variables and toggling them on and off. Here we are just concerned with the state of the machine. Once the machine reaches state=3, then the pattern is completed and can be painted.

Each state represents a different pattern criteria. State starts at zero and if it finds the nine bars that fit the **SetUp (phase)** criteria, it transitions to State 1.

While the machine is in State 1 it looks for the **Intersection** phase. When the **intersection** phase is satisfied the machine moves to State 2.

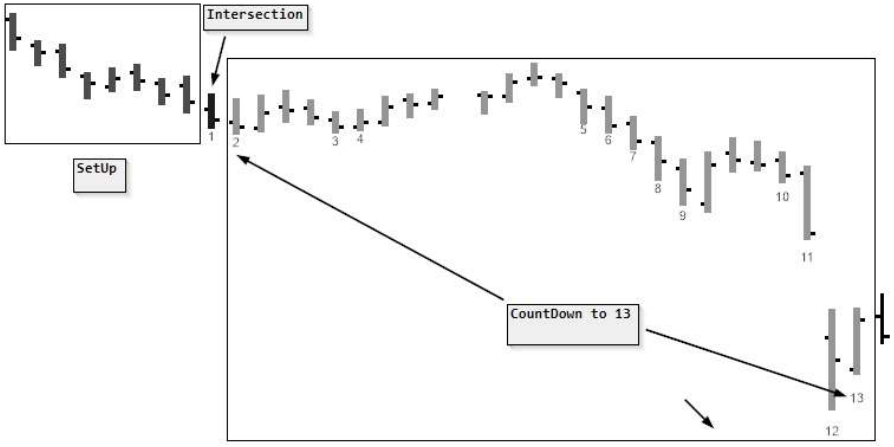
In State 2, the machine looks for the 13 bars that fit the **Countdown** phase. Now this is where, according to a DeMark explanation, that the process can be recycled and the process either reverses to State 1 or State 2. This bit of code takes care of 2 of the 3 conditions for recycling. I didn't fully understand the 3rd condition to recycle so its not included. Here is the recycling code:

```
for index = 0 to 8
begin
      if(close[index] > close[index+4]) then
            checkSum = checkSum + 1;
end;
If checkSum = 9 or h > state1High then
Begin
      state = 0;
      longCountDownCnt = 0;
      PlotPaintBar(High,Low,"Seq.SetUp",cyan);
end;
```

If you have **long SetUp**, and either of these criteria is met then the process is reset. Notice I am using the same functionality from the original code to determine a short **SetUp**. Also if a high price exceeds the highest high during the **SetUp** phase, then the process is recycled or reset

as well. I used a broad brush to program this PaintBar, and there are a plethora of nuances to the Sequential that turns off the CountDown process. This would have made the code a little more complicated, but the same code structure would stay exactly the same.

Here is a screenshot of this PaintBar.



This code can be easily used to create a trading strategy. This FSM (finite state machine) only concerns itself with the three phases of the Sequential Buy Setup. If there is one weakness of the FSM it is that it is usually best to have different machines for the long and short

analysis (double the code). You could combine the two and use bState and sState, but things can get a bit hairy.

Arrays: One of EasyLanguage's Most Scary Words

Arrays are simply just lists and you have been working with them throughout this entire book. The Close price is a special type of array, I refer to it as a Bar Array. It is a list of all of the closing prices right up to the current bar you are testing in a backtest. EasyLanguage allows you to create your own arrays or lists. Unlike the Bar Arrays, it is up to you to manage the indexing of your own arrays. Bar Arrays management is handled internally by TradeStation, so the user doesn't need to concern themselves with anything other than extracting the information they need. If you need the close price from 15 days ago, you now know how to do that: close[15].

This snippet of code will use two arrays to keep track of the day of the week daily True Range ratio to a 30-day ATR measurement taken on the preceding Friday. When you use multiple arrays in most cases you want them to run parallel; meaning that the same element in each different array has some relationship. Here is a simple example:

```
Array: fruit[4](“”),fruitPrice[4](0);
fruit[1] = “orange”;
```

```
fruit[2] = “apple”;
fruit[3] = “peach”;
fruit[4] = “banana”;

fruitPrice[1] = 0.75;
fruitPrice[2] = 0.90;
fruitPrice[3] = 0.76;
fruitPrice[4] = 0.25;
```

Notice how an array is declared? It is a special type of variable so it gets its own declaration statement. You first use the keyword **Array:** and then follow that with the array name and the number of elements that you plan on using in that particular array and then assign a value to each element in the array. In the cases of **fruit** and **fruitPrices** we are defaulting them to empty strings **“”** and **zero** respectively. Here is how you access and print out the values in the arrays using our newly learned **for-loop** construct.

```
for each = 1 to 4
begin
      print(fruit[each],” “,fruitPrice[each]);
end;
```

```
orange    0.75
apple    0.90
peach    0.76
banana    0.25
```

Each (the variable name) starts at 1 and increments to 4 and at each increment a loop is completed. Inside the loop the print statement prints out the values in the **fruit** and **fruitPrices** arrays (like a Sigma equation). You can easily see how the parallel arrays operate:

fruit[1] = orange and fruitPrice[1] = orange price = 0.75

fruit[2] = apple and fruitPrice[2] = apple price = 0.90

And so on...

Now, getting back to our little project. Here is the code and remember just go through it one line at a time:

Using Arrays to Help Record Day Of Week Volatility

```
Vars: volMeasure(1),count(0),dayName(""),DOW(0);
Arrays: binArray[5](0),dayCntArray[5](0);

  { - COMMENTS - NOTICE CURLY BRACKETS
 binArray will keep track of the daily atr ratios
 binArray[1] = Monday
 binArray[2] = Tuesday
 binArray[3] = Wednesday
 binArray[4] = Thursday
 binArray[5] = Friday
 }

if(DayOfWeek(Date) > DayOfWeek(Date of tomorrow)) then
      volMeasure = avgTrueRange(30);
DOW = dayOfWeek(Date);
```

```
binArray[DOW] = binArray[DOW] + TrueRange/volMeasure;
dayCntArray[DOW] = dayCntArray[DOW] + 1;

if(lastBarOnChart) then
begin
      Print(SymbolName,Date:6:0," Day of Week Vol.
Study");
      for cnt = 1 to 5
      begin
            if(cnt = 1) then dayName = "Monday";
            if(cnt = 2) then dayName = "Tuesday";
            if(cnt = 3) then dayName = "Wednesday";
            if(cnt = 4) then dayName = "Thursday";
            if(cnt = 5) then dayName = "Friday";
            Print(dayName,binArray[cnt] /
dayCntArray[cnt]);
          Print(dayName,dayCntArray[cnt]);
      end;
end;
```

And here is the output of the analysis technique when applied to 10-years of ES.D data.

```
@ES.D1210728 Day of Week Volatility Study
Monday            1.09
Monday          473.00
Tuesday           1.03
Tuesday         507.00
Wednesday         1.10
Wednesday       507.00
Thursday          1.13
Thursday        498.00
Friday            1.01
```

```
Friday          494.00
```

What does this tell you? The first number is the number of times the day of week took place over the 10-year period and the following number is the TrueRange of the day of week divided by the 30-Day ATR measure on the prior Friday. You can plainly see the holidays occur more often on Mondays. In respect to volatility, it seems mostly that on average each trading day of the week has the same relative amount of volatility. However, Thursdays seem to be the most volatile. I would have thought Mondays and Fridays would be more volatile; pent up or the need to trade after the weekend and traders covering on Friday.

Each of the two arrays that were used in this code were declared with five elements; one for each day of the week.

```
Arrays: binArray[5](0),dayCntArray[5](0);
```

The **binArray** will hold the accumulated True Range value for each respective day of the week. The **dayCntArray** will accumulate the number of times that the particular day of week occurs in the historic data. This little trick allows us to sample the average true range on Fridays or whatever is the last day of the week, in case Friday is a Holiday.

```
if(DayOfWeek(Date) > DayOfWeek(Date of tomorrow)) then
      volMeasure = avgTrueRange(30);
```

Remember we can peek at tomorrow's date or tomorrow's open. The **DayOfWeek** function returns 1 for Mondays and 5 for Fridays. If **tomorrow** = 1 and **today** = 5, then we know we are sitting on a Friday. If tomorrow = 2 and today =5 then we still know that we are sitting on Friday – there is no Monday in this case. If today = 4 and tomorrow = 1 then we know it is the final day of the week even if it is a Thursday.

Since we will be using the **DayOfWeek** frequently it is assigned to the variable **DOW**. If **DOW** = 1 then **binArray[1] binArray[1] + TrueRange/volMeasure**. This calculation is stored in the array element that corresponds to Mondays. **dayCntArray[1] = dayCntArray[1] + 1**. These two lines are considered accumulators. The first accumulates or sums up all the TrueRange/volMeasure ratios for a particular day. The second line increments the number of Mondays that have been encountered. Notice these two arrays are used in parallel. The final section of the code prints out the information.

```
for cnt = 1 to 5
begin
      if(cnt = 1) then dayName = "Monday";
      if(cnt = 2) then dayName = "Tuesday";
      if(cnt = 3) then dayName = "Wednesday";
```

```
        if(cnt = 4) then dayName = "Thursday";
        if(cnt = 5) then dayName = "Friday";
        Print(dayName,binArray[cnt] / dayCntArray[cnt]);
        Print(dayName,dayCntArray[cnt]);
end;
```

The index in the for loop goes from 1 to 5 and this variable (**cnt**) is used to access the information in the arrays. Now had you used 1 to 6 you would get this error message.

The analysis technique attempted to reference data past the bounds of the array

When you defined the arrays you sized or dimensioned them to only have 5 elements. Here you "stepped out of bonds" by trying to access a 6^{th} element. So arrays are very useful but you must know what the max size you can use and also control the index that is used to index into the arrays (in a later book in this series I will show you how to use a dynamic sized array) to stay within a boundary. Just a quick side bar here. Does this work on this portion of the code?

```
        for count = 0 to 5 //starting at 0 instead of 1
        begin
              if(count = 1) then dayName = "Monday";
              if(count = 2) then dayName = "Tuesday";
              if(count = 3) then dayName = "Wednesday";
              if(count = 4) then dayName = "Thursday";
              if(count = 5) then dayName = "Friday";
        End;
```

It will, because EasyLanguage arrays are 0 **(zero) based**, meaning that the first element can be indexed by 0. So the actual size of an array is N+1, but they can only be indexed up to N. So **dayCantArray[5](0)** can hold six elements, but can only be indexed up to 5. Use **0** (zero) to get that extra element.

I wanted to conclude this book with the concepts of Finite State Machines and arrays, because these concepts are huge cornerstones for further and more complicated research.

What Did Tutorial 13 Teach Us?

- How to create an algorithm to find certain patterns
- Used the pattern finding code to create PaintBars and ShowMes
- The Tale of the Two Paradigms: the two different manners to program multiple step algorithms
- Introduced the concepts of checksums and for-loops
- Programmed the Sequential "Setup" Phase
- Used States to help program very complicated patterns such as Tom DeMark's Sequential
- Introduced the concept of a Finite State Machine and discussed why it is a better approach in many

cases

- Introduced how to properly use Arrays in EasyLanguage and showed how not to be afraid of them
  - Remember arrays in EasyLanguage are 0 based but you do not need to concern yourself with that – just think of it as one extra space you can use.

Appendix A – Source code from each tutorial.

Here is all the source code from the book in one central location.

//EZ_Donchian1_Inputs

```
Inputs: buyLookBack(40),sellLookBack(40);

Buy("DonchBuy") next bar at
      Highest(High, buyLookBack) stop;
SellShort("DonchSell") next bar at
       Lowest(Low, sellLookBack) stop;
```

//EZ_Bollinger1 - with cleaner version

```
inputs:BollingerPrice(Close),length(20),NumDevsDn(2),Num-
DevsUp(2);
variables: LowerBand(0),UpperBand(0),CenterBand(0);


LowerBand = BollingerBand(BollingerPrice,length,-Num-
DevsDn);
```

```
UpperBand =
BollingerBand(BollingerPrice,length,NumDevsUp);
CenterBand = Average(BollingerPrice,length);

// Using the single statement Begin/End is commented out
{If c crosses over UpperBand then
begin
      Buy ("BBandLE") this bar on close;
end;
If c crosses under LowerBand then
begin
      SellShort ("BBandSE") this bar on close;
end;

if marketPosition = 1 then
begin
      If c crosses below CenterBand then
            sell ("LongExit") this bar on close;
end;

if marketPosition = -1 then
begin
      If c crosses above CenterBand then
            buyToCover ("ShortExit") this bar on close;
end;}

//cleaner version
If c crosses over UpperBand then
      Buy ("BBandLE") this bar on close;
If c crosses under LowerBand then
      SellShort ("BBandSE") this bar on close;

if marketPosition = 1 then
      If c crosses below CenterBand then
```

```
            sell ("LongExit") this bar on close;

if marketPosition = -1 then
      If c crosses above CenterBand then
            buyToCover ("ShortExit") this bar on close;
```

//EZ_Keltner1

```
inputs: keltnerLen(20), keltnerNumAtr(1.5),
            profitAmt$(1000), stopLossAmt$(500);

if c crosses above
      keltnerChannel(close,keltnerLen,keltnerNumAtr) then
            buy next bar at open;
if c crosses below
      keltnerChannel(close,keltnerLen,-keltnerNumAtr) then
            sellshort next bar at open;

if marketPosition = 1 then
begin
      If c crosses below
            average(keltnerLen, keltnerNumAtr) then
                  sell this bar on close;
      sell("longLiq") next bar at
            entryPrice - stopLossAmt$ / bigPointvalue
stop;
      sell("longPrf") next bar at
            entryPrice + profitAmt$/bigPointValue limit;
end;

if marketPosition = -1 then
begin
      If c crosses above
            average(keltnerLen,keltnerNumAtr) then
                  buyToCover this bar on close;
      buyToCover("ShortLiq") next bar at
            entryPrice + stopLossAmt$ /bigPointvalue stop;
      buyToCover("ShortPrf") next bar at
            entryPrice - profitAmt$ / bigPointValue limit;
End;
```

//EZ_Keltner2

```
inputs: keltnerLen(20), keltnerNumAtr(1.5),profi-
tAmt$(1000), $stopLossAmt(500);

if c crosses above
      keltnerChannel(close,keltnerLen,keltnerNumAtr) then
            buy next bar at open;
if c crosses below
      keltnerChannel(close,keltnerLen, keltnerNumAtr) then

            sellshort next bar at open;

if marketPosition 1 then
begin
      If c crosses below average(c,keltnerLen) then
            sell this bar on close;
end;
if marketPosition 1 then
begin
      If c crosses above average(c,keltnerLen) then
            buyToCover this bar on close;
End;
setStopLoss(stopLossAmt$);
setProfitTarget(profitAmt$);
```

//EZ_Pyramid1

```
input:trendLen(100),maxPositions(3),profitObj$(500),
maxLoss$(500);
Vars: mp(0);

Mp=marketPosition;
If close>average(c,trendLen) then
Begin
      If c<c[1] and currentShares<maxPositions then
            buy("PyraBuy") next bar at open;
end;
If mp 1 and c>entryPrice+profitObj /bigPointValue then
      sell("Profit") next bar at open;
```

```
If mp 1 and c<entryPrice-maxLoss /bigPointValue then
      sell("Loss") next bar at open;
```

//EZ_Pyramid_Video using same day exit from profit or loss

```
//ES.D 10 years
{Mean  -  100 day moving average
 close < close[1] and c > moving average
 close < close[1] addON up to 3 positions
 profit objective = $500
 stop loss amount  = $500
 long side only }

 inputs:
trendLen(100),maxPositions(3),profitObj$(500),max-
Loss$(500);

 vars: mp(0);

 mp = marketPosition;

 if close > average(close,trendLen) then
 begin
      if close < close[1] and currentContracts < maxPosi-
tions then
            buy("PyraBuy") next bar at open;
 end;

 setStopLoss(maxLoss$);
 setProfitTarget(profitObj$);
```

//EZ_Pyramid2

```
input:
trendLen(100),atrLen(10),maxPositions(2),profitObj$(500),m
axLoss$(500),numShares(100),barsInTrade(20);

vars: mp(0);
```

```
mp = marketPosition;


If close > average(c,trendLen) then
Begin
      If mp = 0 and c < c[1] then
            buy("InitBuy") numShares shares next bar open;
end;

If mp = 1 and barsSinceEntry < barsInTrade and
      currentShares = numShares and
      c < entryPrice - avgTrueRange(atrLen) then
          buy("PyraBetter") numShares shares next bar at
o;

If currentShares = numShares then
      SetStopLoss(Value1);
If currentShares = 2*numShares then
      sell("BreakEven") next bar at entryPrice limit;
If barsSinceEntry > barsInTrade then
      sell("BSESell") this bar on close;

Value1 = maxLoss$;
SetProfitTarget(profitObj$);
```

//EZ_Pyramid3

```
input: trendLen(100),atrLen(10),maxPositions(400),profi-
tObj$(500),
maxLoss$(500),numShares(100),barsInTrade(20);

vars: mp(0),addOn(0),takeOff(0),ATR(0);

mp = marketPosition;

If close > average(c,trendLen) then
Begin
      If mp = 0 and c < c[1] then
            buy("InitBuy") numShares shares next bar at o;
end;
```

```
If mp = 0 then
begin
      addOn = 0;
      takeOff = 0;
      ATR = avgTrueRange(atrLen);
end;

If mp = 1 and barsSinceEntry < barsInTrade and currentSha-
res < maxPositions and
      currentShares = (addOn + 1)* numShares and
      c < entryPrice - (addOn + 1) * ATR then
begin
      addOn = addOn + 1;
      buy("PyraBetter") numShares shares next bar at open;
end;

If mp = 1 and addOn > 0 and
c > entryPrice - addOn * ATR + (takeOff + 1) * ATR/2 then
Begin
      Sell ("TakeOff") numShares shares total next bar at
o;
      takeOff = takeOff+ 1;
end;

If currentShares = numShares then
      SetStopLoss(maxLoss$);
If barsSinceEntry > barsInTrade then
      sell("BSESell") this bar on close;

SetProfitTarget(profitObj$);
```

//MoneyManager

```
Inputs: initCapital(100000),rskAmt(.02); Vars: market-
Risk(0), numContracts(0);

//two methods follow to calculate perceived market risk
//first is commented out
//marketRisk = StdDev(Close,30) * BigPointValue;
marketRisk=avgTrueRange(30) * BigPointValue;
numContracts=(initCapital * rskAmt) / marketRisk;
```

```
Value1=round(numContracts,0);
{Round down to the nearest whole number}
if(value1 > numContracts)
Then
      numContracts=value1-1
else
      numContracts = value1;
numContracts = MaxList(numContracts,1);

Buy("MMBuy") numContracts shares tomorrow at
      Highest(High,40) stop;
SellShort("MMSell") numContracts shares tomorrow at
      Lowest(Low,40) stop;

if(MarketPosition = 1) then
      Sell("LongLiq")next bar at Lowest(Low,20) stop;
if(MarketPosition = -1) then
      BuyToCover("ShortLiq") next bar at

            Highest (High,20) stop;
```

//Money Manager with compounding

```
Inputs: initCapital(100000),rskAmt(.02),maxSize(10);
Vars: marketRisk(0),numContracts(0),workingCapital(0);

marketRisk = StdDev(Close,30)*BigPointValue;

workingCapital = netProfit + initCapital;
numContracts = (workingCapital * rskAmt) /marketRisk;
Value1 = round(numContracts,0);

{Round down to the nearest whole number}
if(value1 > numContracts) then
      numContracts = value1 – 1
else
      numContracts = value1;

numContracts = MaxList(numContracts,1);
```

```
numContracts = Buy("MMBuy") numContracts shares tomorrow
at
      Highest(High,40) stop;
SellShort("MMSell") numContracts shares tomorrow at
      Lowest(Low,40) stop;

if(MarketPosition = 1) then
      Sell("LongLiq")next bar at Lowest(Low,20) stop;

if(MarketPosition = -1) then
      BuyToCover("ShortLiq") next bar at
            Highest(High,20)stop;
```

// EZ_DailyDayTraderCorrect

```
inputs: movAvgLen(100),numDownCloses(3),stopLoss (500);

If numDownCloses = countIf(Close-Close[1],numDownCloses)
and   C > average(c,movAvgLen) then
            buy("DownCAndC Avg") next bar at open;

setStopLoss(stopLoss );

setExitOnClose;
```

//EZ_WilderADX

```
inputs: keltnerLen(20), keltnerNumAtr(1.5), profi-
tAmt(1000), stopLossAmt(500);

if c crosses above
      keltnerChannel(close,keltnerLen,keltnerNumAtr) and
      ADX(20) > 25 then
            buy("KeltBuy") next bar at open;

if c crosses below
      keltnerChannel(close,keltnerLen, keltnerNumAtr) and
      ADX(20) < 25 then
            sellshort("KeltShort") next bar at open;
```

```
if marketPosition 1 then
begin
      If c crosses below average(c,keltnerLen) then
            sell("LongLiq") this bar on close;
      If ADX(20) < 20 and c crosses below average(c,10)
then        sell("LongLiqADX 20") this bar on close;
end;

if marketPosition 1 then
begin
      If c crosses above average(c,keltnerLen) then
            buyToCover("ShortLiq") this bar on close;
      If ADX(20) < 20 and c crosses above average(c,10)
then        sell("ShortLiqADX 20") this bar on close;
end;

//setStopLoss(stopLossAmt);
//setProfitTarget(profitAmt);
```

//EZ_RSIwADX

```
inputs: ADXLen(20), RSILen(20), RSIOBVal(65),
RSIOSVal(35), RSIProfVal(50);



vars: rsiVal(0), adxVal(0), mp(0);

rsiVal = RSI(c,RSILen);
adxVal = ADX(ADXLen);
mp = marketPosition;

If adxVal > 20 and rsiVal crosses above RSIOSVal then
      Buy("RSIADX B") next bar at open;

If adxVal > 20 and rsiVal crosses below RSIOBVal then
      sellShort("RSIADX S") next bar at open;
```

```
If mp = 1 then
Begin
      If rsiVal crosses above RSIProfVal then
            sell("LongLiq 50") next bar at open;
end;
If mp = -1 then Begin
      If rsiVal crosses below RSIProfVal then
            buyToCover("ShortLiq 50") next bar at open;
End;
```

//EZ_Stochastic

```
//EZ_Stochastic
Inputs:stochLen(14),smooth1(3),smooth2(3),StoOBVal(75),
StoOSVal(25),StoLiqVal(50);
Vars: oFastK(0), oFastD(0), oSlowK(0), oSlowD(0),mp(0);

Value1 =
Stochastic(H,L,C,stochLen,smooth1,smooth2,1,oFastK,oFastD,
oSlowK,oSlowD);

mp = marketPosition;

If oFastD crosses above oSlowD and
      oSlowD < stoOSVal then
            buy("Fast>Slow-OS-B") next bar at open;
If oFastD crosses below oSlowD and
      oSlowD > stoOBVal then
            sellShort("Fast<Slow-OB-S") next bar at open;

If marketPosition = 1 then
Begin
      If oSlowD > StoLiqVal then
            sell("StoLongLiq") next bar at open;
```

```
end;
If marketPosition =-1 then
Begin
      If oSlowD < StoLiqVal then
            buyToCover("StoShortLiq") next bar at open;
end;
```

//EZ_LaguerreRSI function

```
inputs: myGamma(numericSimple);

vars: L0(0),L1(0),L2(0),L3(0),CU(0),CD(0),L_RSI(0);

L0 = (1 - myGamma) * close + myGamma*L0[1];
L1 = -gamma * L0 + L0[1] + myGamma * L1[1];
L2 = -myGamma * L1 + L1[1] + myGamma * L2[1];
L3 = -myGamma * L2 + L2[1] + gamma * L3[1];

CU = 0;
CD = 0;

If L0 >= L1 then CU = L0 - L1 Else CD = L1 - L0;
If L1 >= L2 then CU = CU + L1 - L2 Else CD = CD + L2-L1;
If L2 >= L3 then CU = CU + L2 - L3 Else CD = CD + L3 - L2;
If CU + CD > 0 then L_RSI = CU / (CU + CD);

EZ_LaguerreRSI = L_RSI;
```

//EZ_Laguerre Indicator

```
inputs: myGamma(.5);

vars: L_RSI(0);

L_RSI = EZ_LaguerreRSI(myGamma);

SetPlotColor(1,Cyan);
```

```
If L_RSI >= 0.8 then SetPlotColor(1,Red);
If L_RSI <= 0.2 then SetPlotColor(1,Green);

Plot1 (L_RSI, "RSI");
Plot2 (.8);
Plot3 (.2);
```

//EZ_Debug1

```
//This code has a bug
// can you find it?

inputs: atrLen(30),thrust1Mult(0.75),thrust2Mult(1.5);
vars:atr(0);

atr = avgTrueRange(30);

If c < c[1] then
Begin
      buy("BDORB-B") next bar at atr * thrust1Mult stop;
      sellShort("BDORB-S") next bar at atr*thrust2Mult
stop;
end
Else
Begin
      buy("SDORB-B") next bar at atr * thrust1Mult stop;
      sellShort("SDORB-S") next bar at atr*thrust2Mult
stop;
End;
```

//EZ_NRStrategy code for a ShowMe

```
inputs: nrLookBack(7),useTrueRanges(True);
vars: narrowRange(False);

narrowRange = False;
```

```
If useTrueRanges then
Begin
      if TrueRange < Lowest(TrueRange[1],nrLookBack-1)
then
            narrowRange = True;
end
Else
Begin
      If Range < Lowest(Range[1],nrLookBack-1) then
            narrowRange = True;
end;

Value1 = CLOSE ;

{Leave the following as is.  The plot is not named because
there is only one plot, and the default name Plot1 will be
adequate.  The alert does not include a description be-
cause the alerting criteria and the plotting criteria are
the same, and the description will be redundant. }

if NarrowRange then
begin
      Plot1( Value1 ) ;
      Alert ;
End ;
```

//EZ_NRStrategy for PaintBar

```
inputs: nrLookBack(7),useTrueRanges(True);
vars: narrowRange(False);

narrowRange = False;

If useTrueRanges then
Begin
      if TrueRange < Lowest(TrueRange[1],nrLookBack-1)
then
            narrowRange = True;
end
Else
```

```
Begin
      If Range < Lowest(Range[1],nrLookBack-1) then
            narrowRange = True;
end;

//Copy above into a ShowMe study and PaintBar Study
Value1 = HIGH ;
Value2 = LOW ;

{ Leave the following as is.  The plot is not named be-
cause there is only one PaintBar plot - with two sub-plots
- and the default names Plot1, Plot2 will be adequate.
The alert does not include a description because the
alerting criteria and the plotting criteria are the same,
and the description will be redundant. }

if narrowRange then
begin
      PlotPaintBar( Value1, Value2 ) ;
      Alert ;
End ;
```

//EZ_NRStrategy

```
inputs: nrLookBack(7),useTrueRanges(True),numBarsInT-
rade(5);
vars: narrowRange(False);

narrowRange = False;

If useTrueRanges then
Begin
      if TrueRange < Lowest(TrueRange[1],nrLookBack-1)
then
            narrowRange = True;
end
Else
Begin
      If Range < Lowest(Range[1],nrLookBack-1) then
            narrowRange = True;
```

```
end;

//Copy above into a ShowMe study and PaintBar Study

if narrowRange and
      narrowRange[1] then
      buy("RngExplosion") next bar at
            h + minMove/priceScale stop;


If barsSinceEntry > numBarsInTrade then
      sell this bar on close;
```

//EZ_NR_Bollinger Strategy

```
inputs: nrLookBack(7),useTrueRanges(True),BBandLen(20),
            BBandNumStd(2),numBarsInTrade(5);
vars: narrowRange(False),mp(0),upperBand(0),midBand(0);
vars: entryTrigger(0),entryLevel(0);

narrowRange = False;
// we could put the search for a NR in a function
If useTrueRanges then
Begin
      if TrueRange < Lowest(TrueRange[1],nrLookBack-1)
then
            narrowRange = True;
end
Else
Begin
      If Range < Lowest(Range[1],nrLookBack-1) then
            narrowRange = True;
end;

upperBand = bollingerBand(c,BBandLen,BBandNumStd);
midBand = average(c,BBandLen);

If close > upperBand then
      entryTrigger = 1;
```

```
If entryTrigger = 1 then
Begin
      If narrowRange and
      c < upperBand and
      c >  midBand then
      begin
            entryTrigger = 2;
            entryLevel = high + minMove/priceScale;
      end;
end;

If entryTrigger = 2 then
      buy("Trigger=2:B") next bar at entryLevel stop;

mp = marketPosition;

If c <= midBand or mp = 1 then entryTrigger = 0;

if(barsSinceEntry = numBarsInTrade) then sell("TradeExpi-
res") next bar at open;
```

```
[LegacyColorValue = true];
```

{EZ_Sequential Setup
Paint the Sequential Setup Pattern}

```
Vars: index(0),checkSum(0);

checkSum = 0;
for index = 0 to 8
begin
      if(close[index] < close[index+4]) then
            checkSum = checkSum + 1;
end;
if(checkSum = 9)then
begin
      for index = 0 to 8
      begin
```

```
            PlotPaintBar[index](High[index],Low[index],
                  "Sequential",YELLOW);
      end;
end;
checkSum = 0;
for index = 0 to 8
begin
      if(close[index] > close[index+4]) then
            checkSum = checkSum + 1;
end;
if(checkSum = 9)then
begin
      for index = 0 to 8
      begin
            PlotPaintBar[index](High[index],Low[index],
                  "Sequential",RED);
      End;
End;
```

//EZ_TDSequential – broad stroke at pattern

```
Vars: index(0),checkSum(0),
      state(0),state1BarNum(0),
      state1High(0),state2BarNum(0),longCountDownCnt(0);

{Buy SetUp - the setup follows the
 Consecutive bar exact sequence paradigm or CBES followed
 by Variable bar liberal sequence paradigm
 using a State Machine to complet the pattern}

checkSum = 0;
for index = 0 to 8
begin
      if(close[index] < close[index+4]) then
            checkSum = checkSum + 1;
end;

If checkSum = 9 and state = 0 then
Begin
      state = 1;
```

```
      state1BarNum = barNumber;
      state1High = highest(h,9);
      for index = 0 to 8
      begin
            PlotPaintBar[index](High[index],Low[index],
                  "Seq.SetUp",red);
      end;
end;

If state = 1 then // found the setup
Begin
      if high[1] > low[3] or high > low[2] then
      begin
            state = 2;
            state2BarNum = barNumber;
            If state2BarNum = state1BarNum then
                  PlotPaintBar(High,Low,"Seq.SetUp",blue)
            Else
            begin
                  for index = 0 to barNumber –
                              maxList(1,(state1BarNum +
1))
                  begin
                        PlotPaintBar[index](High[index],
                                    Low[index],
                                    "Seq.SetUp",blue);
                  end;
            end;
      end;
end;
If  state = 2 then // intersection found
begin
      if c < l[2] then longCountDownCnt=longCount-
DownCnt+1;
      checkSum = 0;
      for index = 0 to 8
      begin
            if(close[index] > close[index+4]) then
                  checkSum = checkSum + 1;
      end;
      If checkSum = 9 or h > state1High then
      Begin
            state = 0;
```

```
            longCountDownCnt = 0;
            PlotPaintBar(High,Low,"Seq.SetUp",cyan);
      end;
      If longCountDownCnt = 13 then state = 3;
end;

If  state = 3 then //completed the pattern
begin
      for index = 0 to barNumber - (state2BarNum+1)
      begin

            PlotPaintBar[index](High[index],
                  Low[index],"Seq.SetUp",yellow);
      end;
      state = 0;
      longCountDownCnt = 0;
end;
```

//EZ PatternSmasher

```
var: patternTest(""),patternString(""),tempString("");
var: iCnt(0),jCnt(0);
array: patternBitChanger[4](0);

{written by George Pruitt -- copyright 2006 by George Pru-
itt
 This will test a 4 day pattern based on the open to close
 relationship.  A plus represents a close greater than its
open, whereas a minus represents a close less than its
open.  The default pattern is set to pattern 14 +++- (1110
binary).  You can optimize the different patterns by opti-
mizing the patternTests input from 1 to 16 and the orbAmo-
unt from .01 to whatever you like.  Same goes for the hold
days, but in this case you optimize start at zero.  The
LorS input can be optimized from 1 to 2 with 1 being buy
and 2 being sellshort.}

patternString = "";
patternTest = "";

patternBitChanger[0] = 0;
patternBitChanger[1] = 0;
patternBitChanger[2] = 0;
```

```
patternBitChanger[3] = 0;

value1 = patternTests - 1;

if(value1 > 0) then
begin

      if(mod(value1,2) = 1) then
            patternBitChanger[0] = 1;
      value2 = value1 - patternBitChanger[0] * 1;

      if(value2 >= 8) then
      begin
            patternBitChanger[3] = 1;
            value2 = value2 - 8;
      end;

      if(value2 >= 4) then
      begin
            patternBitChanger[2] = 1;
            value2 = value2 - 4;
      end;
      if(value2 = 2) then patternBitChanger[1] = 1;
end;
patternString = "";
for iCnt = 3 downto 0
begin
      if(patternBitChanger[iCnt] = 1) then
      begin
            patternTest = patternTest + "+";
      end
      else
      begin
            patternTest = patternTest + "-";
      end;
end;

for iCnt = 3 downto 0
begin
    if(close[iCnt]> close[iCnt+1]) then
    begin
        patternString = patternString + "+";
    end
```

```
    else
    begin
        patternString = patternString + "-";

    end;
end;

if(barNumber = 1) then
      print(elDateToString(date)," pattern ",patternTest,"
            ",patternTests-1);
if(patternString = patternTest) then
begin
      if (enterNextBarAtOpen) then
      begin
            if(LorS = 2) then
                  SellShort("PatternSell") next bar on
open;
            if(LorS = 1) then
                  buy("PatternBuy") next bar at open;
      end
      else
      begin
            if(LorS = 2) then
                  SellShort("PatternSellBO") next bar at
                  open of tomorrow - avgTrueRange(atrAvg-
Len)
                        * orbAmount stop;
      if(LorS = 1) then
                  buy("PatternBuyBO") next bar at
                  open of tomorrow + avgTrueRange(atrAvg-
Len)
                         * orbAmount stop;
      end;
end;

if(holdDays = 0 ) then setExitonClose;
if(holdDays > 0) then
begin
    if(barsSinceEntry = holdDays and LorS = 2) then
            BuyToCover("xbarLExit") next bar at open;
    if(barsSinceEntry = holdDays and LorS = 1) then
            Sell("xbarSExit") next bar at open;
End;
```

Appendix B – Video Links via Vimeo.

Tutorial 1 –
https://vimeo.com/578505056/9cb1cfafb7
Tutorial 2 –
https://vimeo.com/578168730/74d646c772
Tutorial 3 -
https://vimeo.com/578272185/2bfc2d3b38
Tutorial 4 -
https://vimeo.com/578577571/2822d8820c
Tutorial 5 -
https://vimeo.com/578721322/57953c9b2e
Tutorial 6 -
https://vimeo.com/579520861/f3649563e2
Tutorial 7 -
https://vimeo.com/580064016/8e6b1bc7a7
Tutorial 8 -
https://vimeo.com/580090161/d362ba2e9b

Tutorial 9 -
https://vimeo.com/580747251/c45cc7730f
Tutorial10 –
https://vimeo.com/581252841/573f62c917
Last Tutorial -
https://vimeo.com/582109515/08cc067abd