

Important Information and Disclosures:

TradeStation seeks to serve institutional and active traders. Please be advised that active trading is generally not appropriate for someone of limited resources, limited investment or trading experience, or low risk tolerance, or who is not willing to risk at least \$50,000 of capital.

This book may discuss in detail how TradeStation is designed to help you develop, test and implement trading strategies. However, TradeStation does not provide or suggest trading strategies. We offer you unique tools to help you design your own strategies and look at how they could have performed in the past. While we believe this is very valuable information, we caution you that simulated past performance of a trading strategy is no guarantee of its future performance or success. We also do not recommend or solicit the purchase or sale of any particular foreign currencies, commodities, securities or derivative products. Any symbols referenced in this book are used only for the purposes of demonstration, as an example, not a recommendation.

Finally, this book may discuss automated buy and sell alerts and/or electronic order placement and execution. Please note that even though TradeStation has been designed to automate your trading strategies and deliver timely order placement, routing and execution, these things, as well as access to the system itself, may at times be delayed or even fail due to market volatility, quote delays, system and software errors, Internet traffic, outages and other factors.

All proprietary technology in TradeStation is owned by TradeStation Technologies, Inc., an affiliate of TradeStation Securities Inc. All other features and functions of TradeStation are provided directly by TradeStation Technologies. TradeStation® and EasyLanguage® are registered trademarks of TradeStation Technologies, Inc. "TradeStation," as used in this document, should be understood in the foregoing context.

The price charts presented in this book are for demonstration purposes only and are not a recommendation or endorsement of the symbols or analysis displayed within the charts.

PAST PERFORMANCE IS NOT NECESSARILY INDICATIVE OF FUTURE RESULTS.

Copyright © 2002-2013 TradeStation Technologies, Inc. All rights reserved. Licensed to its affiliate, TradeStation Securities, Inc. (Member NASD, NYSE, SIPC, and NFA)

Second Edition
June 2012

Table of Contents

About This Book	v
1. What is EasyLanguage?.....	1
2. The TradeStation Development Environment (TDE).....	3
The EasyLanguage Editor	4
The EasyLanguage Dictionary.....	5
Default Properties Settings for EasyLanguage Documents.....	8
3. Understanding Bars in a Chart.....	9
Note: MaxBarsBack	9
4. EasyLanguage Elements – Words, Statements, Expressions and Punctuation.....	11
EasyLanguage Words	11
EasyLanguage Expressions and Statements	12
EasyLanguage Punctuation.....	13
5. Plot Statements.....	15
6. Indicators.....	17
Exercise: *01 The Close.....	18
7. Setting Properties for EasyLanguage Documents	21
Exercise: *02 Volume.....	23
Challenge 1: *03 High and Low.....	25
8. Mathematical Operators	27
Exercise: *04 Real Body	28
9. Order of Operations.....	29
Exercise: *05 MidPrice	30
Exercise: *06 VolWtdRange.....	32
10. Referencing Data from Previous Bars	33
Exercise: *07 NetChangeOsc	34
Challenge 2: *08 Bands	35
11. Numeric Variables	37
User-declared Numeric Variables.....	37
Variable Declaration Statement	37
Variable Assignment Statement	37
Exercise: *09 Bands2.....	38
Exercise: *10 NetChangeOsc2	40
Pre-declared Numeric Variables.....	41
12. Using Functions.....	43
Exercise: *11 Momentum	44
Exercise: *12 Real Body Avg	46
Challenge 3: *13 Envelope.....	48
13. Inputs.....	49
Exercise: *14 Envelope2	51
Exercise: *15 Trailing Hi Lo.....	55
14. Relational Operators	59
15. If...Then Statements	61
16. Writing Alerts in EasyLanguage.....	63
Exercise: *16 Envelope2 Alert.....	64
Exercise: *17 Mov Avg & Bands	67
17. ShowMe Studies	71
18. Block If...Then Statements	73
Exercise: *18 Wide Range	74

19. If...Then...Else and NoPlot Statements	77
Challenge 4: *19 Weak Close	78
20. PaintBar Studies	79
Exercise: *20 MomentumPositive	79
21. Referencing Bar Date in EasyLanguage	81
Exercise: *21 MyDay	82
22. Referencing Bar Time in EasyLanguage	83
23. Logical Operators	84
Challenge 5: *22 Parts Of Day	85
24. Strategies	87
EasyLanguage Order Syntax for Strategies	87
Exercise: *23 Breakout	89
Exercise: *24 Mov Avg Cross	95
25. True/False Variables	99
User-declared True/False Variables	99
Variable Declaration Statement	99
Variable Assignment Statement	99
Exercise: *25 Momentum Cross	100
Challenge 6: *26 Key Reversal	103
Pre-declared True/False Variables	104
26. Built-in Stops	105
Exercise: *27 Breakout2	106
Exercise: *28 RSI OB_OS	108
27. Multidata: Referencing Multiple Data Sets	111
Exercise: *29 Multidata MA	112
28. Dynamic Formatting	115
Exercise: *30 High Volume Bars	116
29. Color Gradients	117
30. Plot Statements for RadarScreen	119
Exercise: *31 Real Body Avg RS	120
31. Variables: Capturing Values	125
Exercise: *32 Intraday H and L	126
32. Variables: Counting Events	129
Exercise: *33 Streak	130
Exercise: *34 Trade Highs Lows	134
33. Viewing Output from EasyLanguage: Print Log	137
34. Viewing Output from EasyLanguage: Analysis Commentary	138
Exercise: *35 Momentum Cross2	139
Appendix A	143
TradeStation Development Environment	143
Appendix B	149
Volume and Ticks	149
Appendix C	153
Attending TradeStation Live In-Person Training Courses	153
Appendix D	155
Importing the EasyLanguage Examples for this Course	155
Course Challenge Videos	155
Appendix E: Course Challenge Answers	157

About This Book

Thank you for purchasing the *EasyLanguage Home Study Course, Part I*. This self-paced course is based on the TradeStation two-day, live hands-on training course, *EasyLanguage Boot Camp*.

The goal is simple: to become fluent in working with TradeStation's revolutionary trading language, EasyLanguage. Achieving fluency means not only being able to translate your own trading ideas into analysis techniques and strategies in EasyLanguage, but also being able to read, understand, and learn from what others have written. You will accomplish this by creating and examining 35 separate EasyLanguage studies by the end of the course. These include indicators, ShowMe and PaintBar studies and, of course, rule-based trading strategies.

Throughout the book, a simple format will be followed. First, a particular element or ability of EasyLanguage will be examined and explained. Second, the book will detail several sample **Exercises** regarding the chapter topic. Periodically, you will be assigned a related **Challenge** to complete on your own. Just like our live course, this book is designed to be hands-on, so you're expected to work through the **Exercises** and **Challenges** right at your computer using TradeStation.

As with EasyLanguage generally, all the exercises in the course are applicable to stocks, futures and forex markets, regardless of the illustrations that may appear in the book. Feel free to insert the completed exercises into any chart, particularly the markets you follow most closely. This will reinforce the value of knowing EasyLanguage and prompt you to come up with new ideas to try on your own.

Challenge answers are provided in the appendix. **As an added bonus, each Challenge answer may be viewed as its own online movie with audio narration at:**
<http://www.tradestation.com/ELHSC>

That web page also contains all of the EasyLanguage for all of the studies created in this course as one EasyLanguage document (or ELD) file, which can be imported directly into the TradeStation Platform on your computer.

It should be stressed that this book is not a "how-to" use TradeStation. It is recommended that you have an understanding and familiarity with TradeStation before beginning this course. This includes such tasks as creating and managing Chart Analysis windows, as well as general file, window, workspace, and desktop management skills. An excellent place to start would be with TradeStation's online tutorials, which reside on the TradeStation Support Center, and may be accessed from the *Help* menu inside the TradeStation Platform or at <https://www.tradestation.com/support>.

You may have purchased this course to use as a refresher after attending a live EasyLanguage course or perhaps because you haven't yet been able to attend in person. Whatever the reason, you now possess a course designed to teach you how to use TradeStation's EasyLanguage to write your own custom analysis techniques and trading strategies that you can then test and automate.

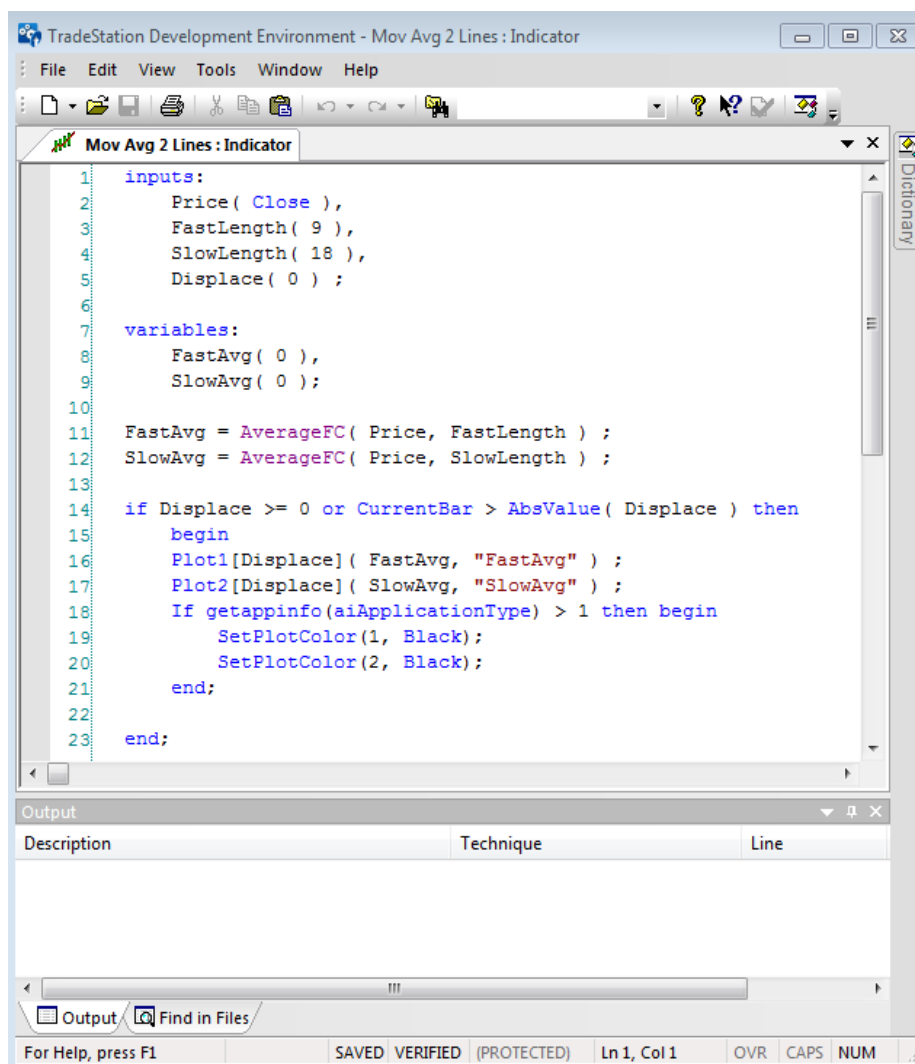
Thank you,

The TradeStation Team

1. What is EasyLanguage?

EasyLanguage is a programming language for traders; it is made up of words, operators, and punctuation used to create indicators and trading strategies based on rules and instructions using market data.

It is true that EasyLanguage is designed to use simple English-like terms that one trader would use to describe a trading idea to another trader; however it is a programming language. This means there are certain rules and guidelines we must follow to ensure that the EasyLanguage studies we create can be understood by and applied within TradeStation. Here's a tip: don't fight the rules of EasyLanguage. If it is required to use a semicolon or parentheses in a certain place, do it. Accepting the rules of the language regarding grammar, statement structure, and punctuation will reduce the learning curve and greatly facilitate the task at hand. In addition, creating good EasyLanguage habits from the start will make getting through the exercises presented in this course that much easier.



```
1  inputs:
2      Price( Close ),
3      FastLength( 9 ),
4      SlowLength( 18 ),
5      Displace( 0 ) ;
6
7  variables:
8      FastAvg( 0 ),
9      SlowAvg( 0 );
10
11  FastAvg = AverageFC( Price, FastLength ) ;
12  SlowAvg = AverageFC( Price, SlowLength ) ;
13
14  if Displace >= 0 or CurrentBar > AbsValue( Displace ) then
15  begin
16      Plot1[Displace]( FastAvg, "FastAvg" ) ;
17      Plot2[Displace]( SlowAvg, "SlowAvg" ) ;
18      If getappinfo(aiApplicationType) > 1 then begin
19          SetPlotColor(1, Black);
20          SetPlotColor(2, Black);
21      end;
22  end;
23  end;
```

The screenshot shows the TradeStation Development Environment (TDE) window titled "Mov Avg 2 Lines : Indicator". The main editor displays the EasyLanguage script for this indicator. The script defines inputs (Price, FastLength, SlowLength, Displace), variables (FastAvg, SlowAvg), and logic for calculating moving averages and plotting them. The bottom of the window features an "Output" pane with columns for Description, Technique, and Line, and a status bar at the very bottom showing "For Help, press F1", "SAVED VERIFIED (PROTECTED)", and "Ln 1, Col 1".

Here is a typical indicator written in the TradeStation Development Environment (TDE).

2. The TradeStation Development Environment (TDE)

Whenever you create or modify EasyLanguage studies in TradeStation, you'll be working in the TradeStation Development Environment (referred to as the Development Environment or, simply, the TDE). The TDE opens as a separate application on your computer. Open the TDE from the TradeStation Platform by clicking on the EasyLanguage icon in the Tools Section of the Shortcut Bar. The TDE includes the EasyLanguage Editor and additional utilities for aiding you in working with EasyLanguage.

TradeStation Development Environment Features:

- **Programming Editor** –A full-featured word-processing like editor for creating and modifying EasyLanguage instructions that lets you communicate your trading ideas to TradeStation. New analysis techniques may be created or existing ones may be modified, including indicators, ShowMe and PaintBar studies, and trading strategies. The Editor also includes checks for proper syntax and grammar.
- **Dictionary** – A reference dictionary of EasyLanguage words, functions, etc.
- **Output Bar** – A desktop bar that returns information about the EasyLanguage document you are working on.
- **Syntax coloring for EasyLanguage** – EasyLanguage “parts of speech” such as functions and reserved words are color-coded automatically as they are typed.
- **Default properties settings** –Analysis techniques and strategies may be assigned default properties for display, such as color and style settings.
- **AutoComplete** – Auto complete is a technology in the EasyLanguage Editor that monitors your typing and will prompt you with a pop-up window displaying a list of reserved words and functions that could be used to complete your entry.
- **Outlining** – Outlining is a feature in the editor that allows you to collapse and expand blocks of code.

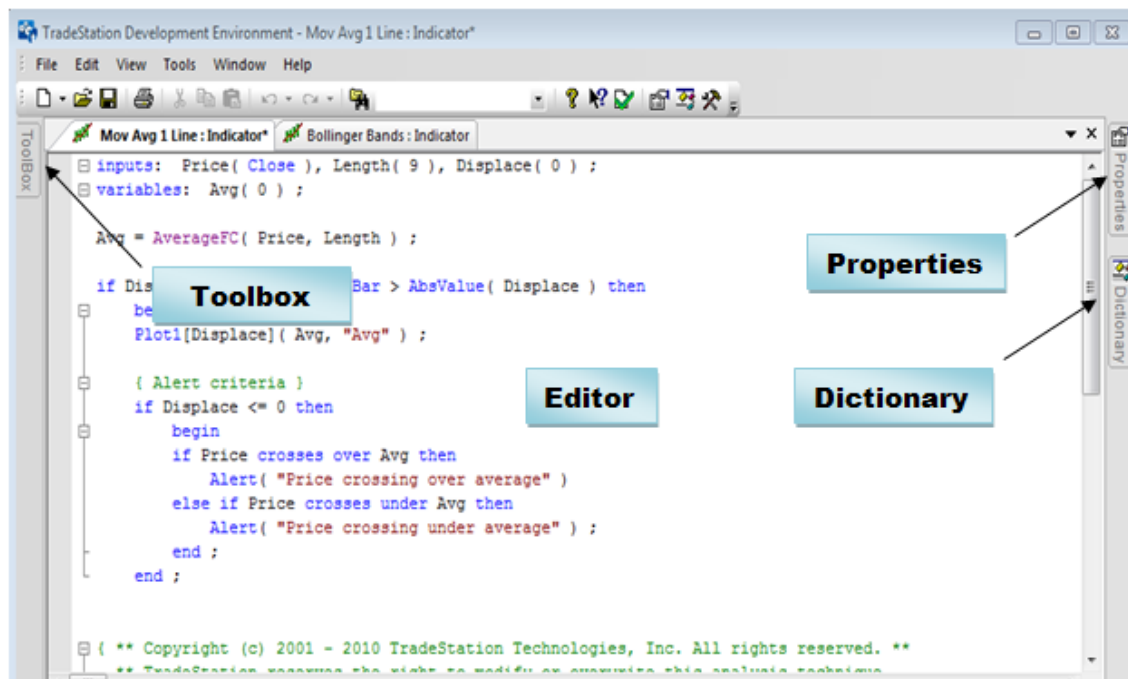
Tip: You may want to create a shortcut to the Development Environment and place it on your Windows desktop.

The EasyLanguage Editor

The EasyLanguage Editor opens existing EasyLanguage documents for review and editing and also allows you to create new analyses and strategies from scratch. Each document opened in the Editor is tabbed at the top of the Development Environment. Documents may be displayed as windows instead of tabs by unchecking Arrange in tabs on the Window menu.


For easier editing, line numbers may be displayed in the Editor by following the Tools – Options menu sequence and checking Line Numbers on the General tab of the Options dialog.

An EasyLanguage study or strategy must be **Verified** before it can be used in the TradeStation Platform. Verification checks the grammar and syntax of the document to see that it conforms to EasyLanguage requirements. Verification may be done from the File menu, the Verify toolbar button or the F3 key on your keyboard. Verification also saves your document. You can check the verified state of a study in the lower right status bar.



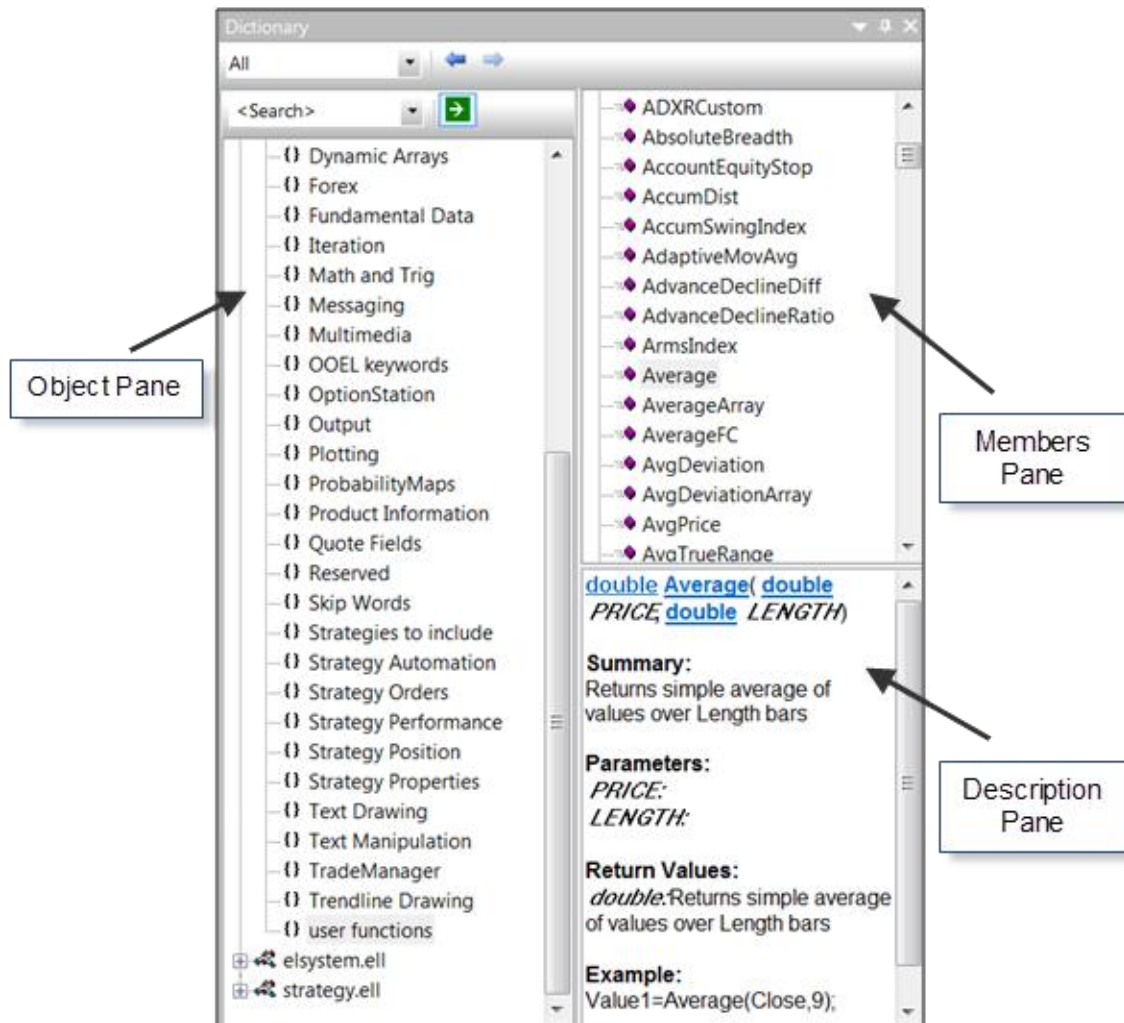
Note: The Toolbox and Properties sliders access advanced EasyLanguage features not discussed in this book.

The EasyLanguage Dictionary


The EasyLanguage Dictionary contains all of the reserved words and functions that may be used when creating EasyLanguage studies. It may be accessed by hovering or clicking on the Dictionary tab at the right side of the Development Environment, from the View menu or by clicking on the Dictionary icon on the toolbar . This is an extremely helpful resource that will save you from having to remember hundreds of reserved words and functions.

The Dictionary has 3 panes. The Objects pane to the left is organized by category so it is easy to find the words needed to express your trading ideas. Click on a category and its members are displayed to the right, in the Members pane. Click on a category in the Objects pane, or an item in the Members pane, and helpful information is displayed in the Description pane at the lower right. The Description pane will often include links to other useful information.

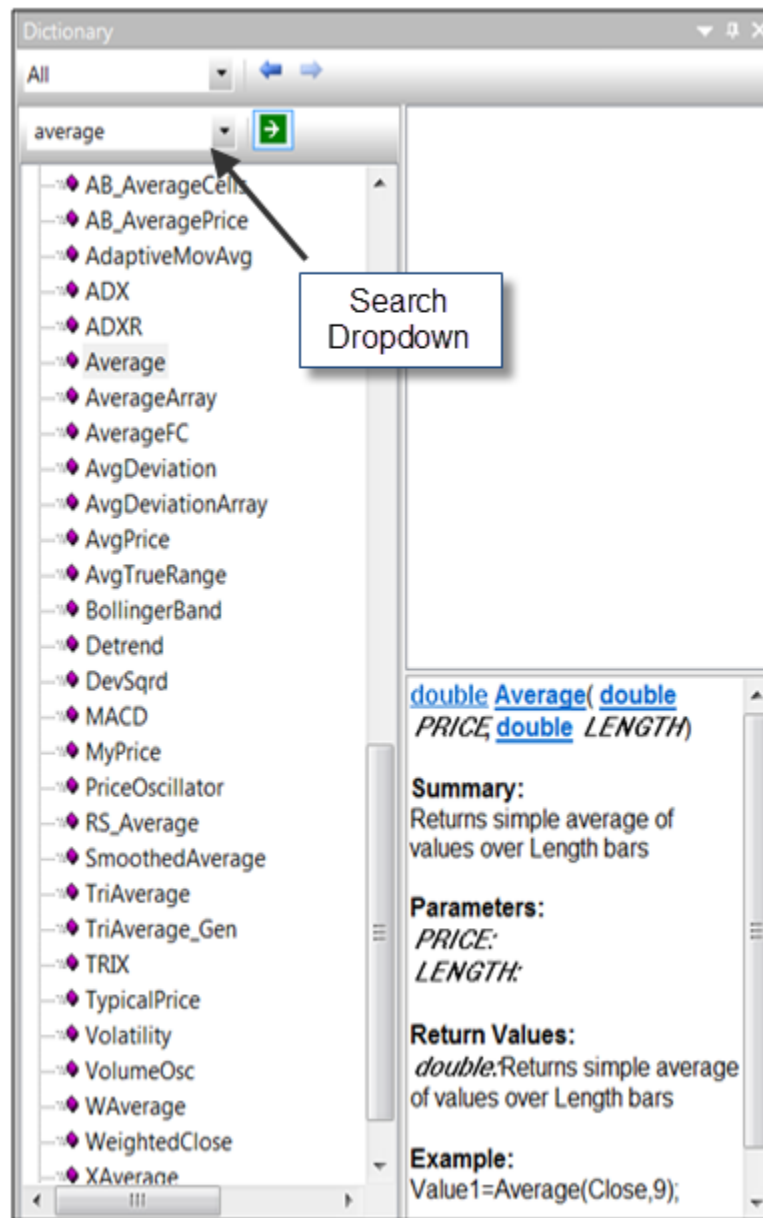
For example, the following screenshot shows the category *easylanguage.ell - User Functions* selected in the *Objects* pane on the left and those *User Functions* listed in the *Members* pane to the right. In this example, the function *Average* is selected.



In addition, you can search for specific words using the *Search* dropdown above the Objects pane. Using this feature, you can search for an EasyLanguage word or function containing the

specified search criteria, then drag and drop it directly into your analysis technique or strategy. To use *Search*, type the word or part of the word you are searching for in the *Search* dropdown then click the arrow  in the green box to the right of the dropdown.

The results will appear in the Objects pane beneath the Search dropdown.



Output

The Output bar may be accessed by hovering or clicking on the Output tab at the bottom of the Development Environment. The *Output* tab displays any syntax errors found when verifying an analysis technique or strategy. This helps you to find errors easily and quickly so you can resolve them. Verification ensures the EasyLanguage commands used to create an analysis technique or strategy follow the EasyLanguage syntax rules. If no errors are found, the tab will display *0 error(s)*, *0 warning(s)* when your analysis technique or strategy is verified successfully. During the verification process, the analysis technique or strategy is also saved. If the analysis technique or strategy that you are verifying is currently applied somewhere in TradeStation, a Chart Analysis window or RadarScreen Window, it will be automatically recalculated after verification.

Syntax Coloring

Syntax coloring allows you to identify specific EasyLanguage word categories such as reserved words, functions, text, and others using specific color settings for each category. This feature is extremely useful identifying problems such as spelling and syntax errors prior to verification. EasyLanguage documents use built-in color schemes by default.

The default syntax-coloring scheme is as follows:

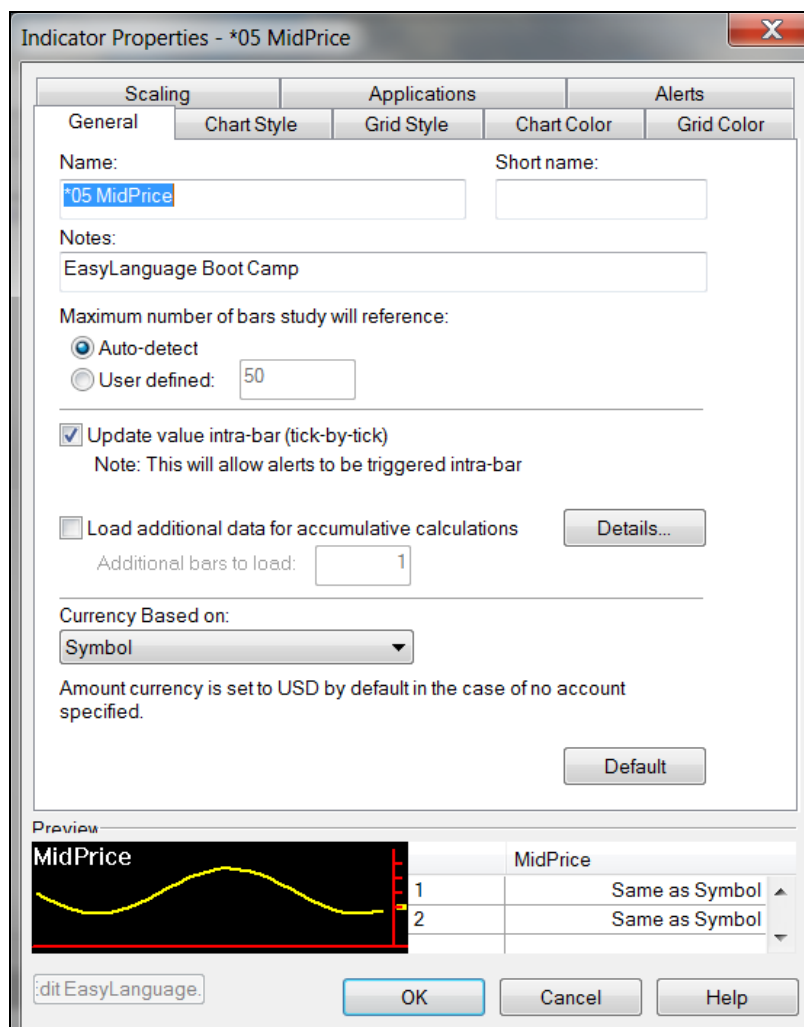
EasyLanguage Category	Default Color
Attributes	Dark Gray
Comments	Green
Functions	Purple
General Statements & Instructions	Black
Quote Fields	Blue
Reserved Words	Blue
Skip Words	Green
String Text	Dark Red

Default Properties Settings for EasyLanguage Documents

The Properties settings for EasyLanguage studies allow you to customize and set defaults for specific display or calculation elements of either existing or newly created EasyLanguage studies.

The figure below shows a *Properties* dialog for an indicator. This may be accessed by right-clicking in the Editor window of a verified document and selecting *Properties* from the shortcut menu, or from the File menu. As you can see, there are tabs for items such as style, color and alerts. As we work through this book creating each exercise, we'll explore many of the tabs contained in the dialog.

Note: Always Verify successfully before attempting to set properties for a new study.



Note: Properties settings may be managed from either the Development Environment or from the window in which the analysis technique or strategy is being used, for example, a Chart Analysis or RadarScreen window.

3. Understanding Bars in a Chart

As we work with EasyLanguage, one of the basic principles to remember is that TradeStation thinks in bars (not ticks, minutes, or days); more specifically, it is the bars on a chart that make the analysis.

TradeStation processes the EasyLanguage instructions, top to bottom, for each bar on the chart, working across the chart, bar by bar, from left to right, after satisfying the Maximum Number of Bars Study will Reference (MaxBarsBack).

Note: MaxBarsBack – Every analysis technique that references historical data requires a certain number of historical bars before it can start calculating. This required starting data is called MaxBarsBack; the maximum number of bars back the study will need in order to calculate the first time in the chart.

Each bar on a chart contains a certain number of data points which can be referenced by EasyLanguage studies.

The following is a list of the data points contained in each bar of a chart depending on the type of symbol and interval:

- Open
- High
- Low
- Close
- Date
- Time
- Volume
 - Volume on Up Ticks
 - Volume on Down Ticks
- Ticks
 - Number of Up Ticks
 - Number of Down Ticks
- Open interest

Also available via EasyLanguage (depending on type of symbol and interval):

- Options-related data:
 - Implied volatility
 - Option volume (Puts and calls)
 - Option open interest (Puts and calls)
- Fundamental data
- Commitments of Traders

4. EasyLanguage Elements – Words, Statements, Expressions and Punctuation

EasyLanguage Words

Like any other language, EasyLanguage is made up of words. EasyLanguage words generally fall into five categories:

- **Reserved Words** – Reserved words are exactly that, reserved by EasyLanguage with a pre-defined meaning. These would be words like `Open`, `Close`, `Plot1` and `Buy`. For example, the reserved word `High` means the high of a bar.
- **Functions** – These are words that call a stored formula into an EasyLanguage document. Words such as `Average`, `RSI`, and `Lowest` are all EasyLanguage functions.
- **User-defined Words** – These words are really a broad category for words that you will create as you write EasyLanguage studies. Some of the words you will create will be used as plot names, inputs, and variables.
- **Skip Words** – These words are used to improve readability, but are skipped over by TradeStation when executing EasyLanguage instructions. These would include words such as `of`, `the`, and `at`.
- **Attributes** – These are words used to set an operating or calculation rule for an analysis technique or strategy. Words like `LegacyColorValue` and `IntraBarOrderGeneration` are attributes.

Note: EasyLanguage is not case sensitive. Keep in mind however, using both upper and lower case when creating EasyLanguage studies can greatly improve readability when referring back to or checking your work.

EasyLanguage Expressions and Statements

Words are the basic building blocks of any language. However, to become fluent in a language requires properly using groups of words to form expressions and statements. All EasyLanguage instructions or rules consist of statements, which are like sentences in spoken language. Sentences can express one simple thought or a series of thoughts.

The two most common types of EasyLanguage expressions are:

- **Numeric expressions** refer to, or calculate, a numeric value. For example, the reserved word `High` is also a numeric expression because it refers to some numeric value. In addition, `Volume * 2` and `(High-Low) * .5` are numeric expressions.
- **True/False expressions** compare two numeric values and are either `TRUE` or `FALSE`. For example:

<code>Close = Open</code>	The Close equals the Open.
---------------------------	----------------------------

<code>High > High[1]</code>	The High is greater than the High of 1 bar ago.
--------------------------------	---

<code>Date <> Date[1]</code>	The Date is not equal to the Date of 1 bar ago.
------------------------------------	---

Four common types of EasyLanguage statements are:

- *Input declaration* statements
- *Variable declaration* statements
- *Variable assignment* statements
- For analysis techniques: *Plot* statements

Statement Sequence

As a general guideline, EasyLanguage statements should appear in this order before verification:

- Input declaration statement
- Variable declaration statement
- Variable assignment statement(s)
- For analysis techniques: `Plot` statement(s)
- For strategies: `Buy` and `Sell` statement(s)

Note: There are instances when statements are intentionally written in other sequences.

EasyLanguage Punctuation

A critical step in becoming fluent in EasyLanguage is to gain a thorough understanding of its punctuation. Below is a reference to the most common uses of EasyLanguage punctuation. Throughout this book all of these types of punctuation will be used as you create EasyLanguage studies.

<code>;</code>	Semi-colon — Denotes the end of each unique EasyLanguage statement.
<code>()</code>	Parentheses — Used to group parameters or for controlling mathematical operations.
<code>,</code>	Comma — Separates items in a list.
<code>[]</code>	Square brackets — Used to reference data from a previous bar, or to displace a plot, or used to access elements within an array.
<code>" "</code>	Quotes — Denote a text or label (e.g., <code>"plot name"</code>).
<code>:</code>	Colon — Denotes the declaration of a list.
<code>{ }</code>	Curly brackets — Any text between curly brackets is notation (comments) that is not calculated as part of the EasyLanguage instructions.
<code>//</code>	Double forward slash — Any text following a double forward slash, for the remainder of that line only, is notation (comments) and is not calculated as part of the EasyLanguage instructions.

Important: Line returns, line spacing, and paragraph indents are ignored by EasyLanguage and are generally used for improved readability. They do not affect the EasyLanguage instructions.

5. Plot Statements

Plot statements instruct TradeStation where to draw plots in a Chart Analysis window, or what to place in a cell in a RadarScreen or OptionStation window. They are used for TradeStation indicators, ShowMe and PaintBar studies but are not used in strategies. An analysis technique may contain a maximum of 99 simultaneous plot statements.

If, for example, you were creating an indicator and wanted this indicator to plot a line on a chart, how would you tell TradeStation to plot this line? You can't simply type, "TradeStation, please plot this line." Doing it with EasyLanguage is just a bit different.

The structure of a plot statement for an indicator in EasyLanguage is:

```
PlotN(numeric expression, "plot name") ;
```

where $N = 1$ to 99.

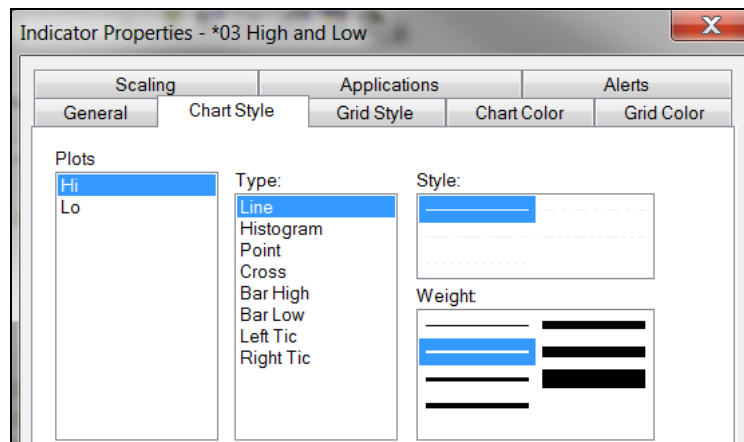
An example of a simple plot statement would be:

```
Plot1(Open, "The Open") ;
```

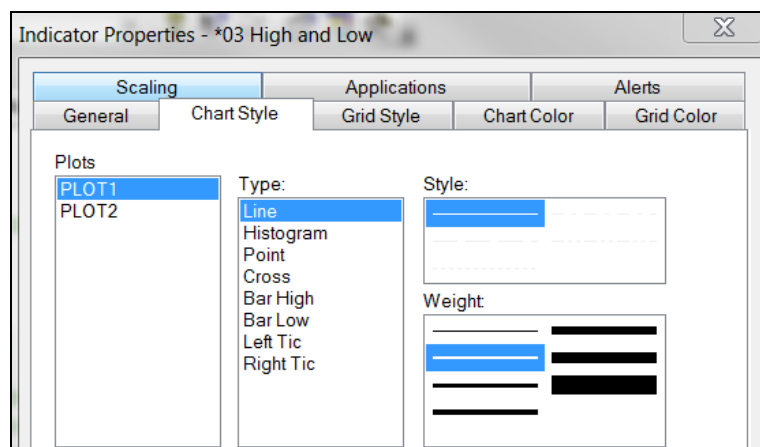
The word `Plot1` in the sample plot statement above states that this is the first plot for this analysis technique.

The *numeric expression* to be plotted, `Open` in the sample statement above, can be thought of as what value is to be plotted, or where on the y-axis of the chart the plot is to appear. In this example, TradeStation is being instructed to place a plot at the numeric value that represents the `Open` for each bar.

"The Open" is the *plot name* and is optional when writing a plot statement. However, it is recommended that the plot name be included for each plot statement created. Why? When modifying analysis techniques with multiple plot statements for items like style and color, it is easier to identify specific plots if they are named. Look at the difference. The screenshot below shows the *Indicator Properties* dialog box for an indicator that contains two plots, both of which are named. Notice the names *Hi* and *Lo* in the *Plots* section of the dialog.



The following screenshot shows the *Indicator Properties* dialog box for the same indicator with the plot names omitted in the EasyLanguage. This results in the plots appearing with the names *PLOT1* and *PLOT2*, instead of the more identifiable names *Hi* and *Lo* as above.



Plot names will also appear in the Data Tip and the Data Window of the Chart Analysis window, making it easy to find the value of a particular plot when examining the chart.

6. Indicators

TradeStation comes supplied with dozens of analysis techniques and strategies including indicators, ShowMe, and PaintBar studies. All of these, of course, are written in EasyLanguage. In addition, TradeStation is an open development Platform, allowing you to view (or modify if you choose) any pre-existing analysis techniques by simply opening them in the Development Environment. This provides users with a great way to learn by seeing how other studies are written. By viewing pre-written EasyLanguage studies in TradeStation you may

- Gain an understanding of how and why a certain study behaves the way it does
- Reduce the EasyLanguage learning curve
- Generate additional analysis ideas

Tip: Should you decide to modify any existing analysis techniques, it is always best to save the changes with a different name, thereby preserving the original file for later reference and use.

In this course, we will be creating EasyLanguage examples from scratch, starting with indicators. An *indicator* in TradeStation may be defined as a mathematical calculation using the data from each bar, and then plotting the calculated value for each bar on the chart. An indicator is generally plotted on a chart as either a line or histogram, or combination of the two.

To begin, you should open the TradeStation Platform.

Now, click the *EasyLanguage* icon in the Tools group of the Shortcut Bar. This will open the TradeStation Development Environment.



In the TradeStation Development Environment, use the File – New – Indicator menu sequence, or the new document dropdown toolbar button, to begin creating our first new indicator.

Exercise: *01 The Close

Learning objective: Using plot statements in indicators.

Description: This indicator plots a line connecting the closing prices of the bars in a chart.

In the *New Indicator* dialog box, type **#01 The Close** in the *Name* field to name this indicator.

Note: The name for each EasyLanguage study **you** create in this course will begin with a pound sign (#) followed by a number and a word or two describing the analysis technique; for example, **#01 The Close**. Notice that each corresponding EasyLanguage study listed in this book begins with an asterisk (*) and NOT the pound sign (#). The purpose here is as follows. All the EasyLanguage for all of the studies created in this book can be found at:

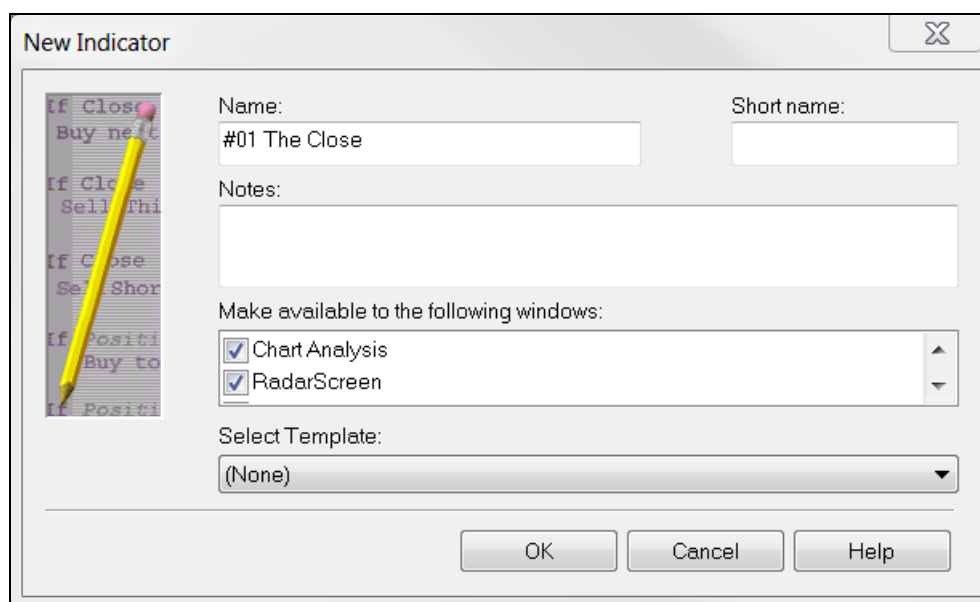
<http://www.tradestation.com/ELHSC>

As mentioned earlier, this is all contained in one EasyLanguage document (or ELD) file, and these studies can be imported directly into your TradeStation Platform. Prefacing the study names which you create throughout this book with a pound sign (#) will enable you to distinguish them from those imported from the web page for this course (denoted with an asterisk).

After this course, as you begin to write your own analysis techniques and strategies, it may be useful to create your own naming system, using it consistently for any EasyLanguage documents you create. Perhaps you will use your initials followed by an underscore to begin the name of each study. This will simplify the access and identification of those documents you create, and help distinguish them from those studies that are supplied by TradeStation or those you have added from 3rd party providers.

In the *Select Template* dropdown menu, select *(None)* from the list provided.

The *New Indicator* dialog should now resemble the screenshot below.



Tip: When creating analysis techniques, the Development Environment provides a variety of preformatted templates to assist you in your efforts. These templates can be extremely helpful in understanding how various analysis techniques should be formatted and structured. When you have some additional time, it is recommended that you experiment with them. However, for the purposes of this course, we will begin each task with a blank Editor window.

Clicking the *OK* button will create a blank window for a new indicator. To create **#01 The Close** indicator, type:

```
Plot1(Close, "Close");
```

You probably noticed several on-screen prompts while you were typing this statement. As you typed the word *Plot*, the Editor's auto complete feature generated a list of EasyLanguage words that matched what you were typing. You may finish typing the word or, if you find it in the auto complete box, you may double-click on it or press the enter key to complete it automatically.


Additionally, when you typed the left parenthesis, auto complete displayed a tool tip reminding you of additional information required for a reserved word or function. We will have a closer look at these later when we begin using functions.

Let's examine this indicator:

Plot1 tells TradeStation that this is the first plot for this analysis technique. Remember, an EasyLanguage study (indicator, ShowMe or PaintBar) may have up to 99 different plots.

Close is the numeric expression (and a reserved word) to be plotted and instructs TradeStation to place a plot at the closing price of each bar on the chart.

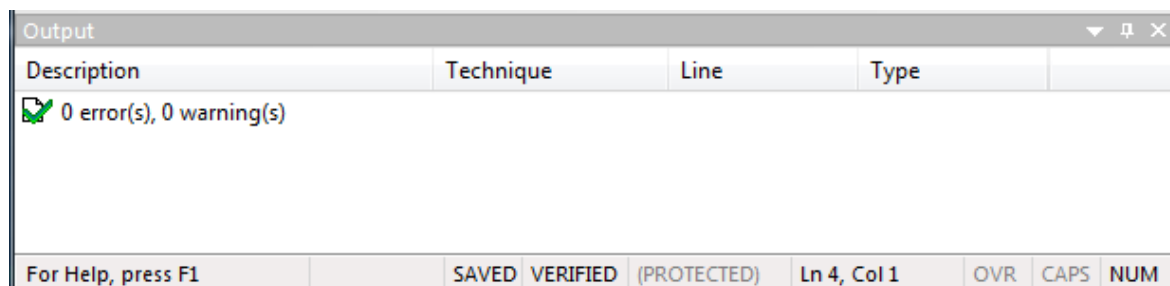
"*Close*" is the plot name and is preceded and separated from the numeric expression, *Close*, by a comma. The numeric expression and plot name sit within a set of parentheses and the entire plot statement is completed with a semi-colon, as are all EasyLanguage statements.

To check your work, you will need to verify it. Do this by using the *Verify* button located on the toolbar  or by right-clicking in the Editor window and selecting *Verify* from the shortcut menu. You may also use the *F3* key on your computer keyboard. If you've typed the EasyLanguage correctly, you should see the message VERIFIED, in dark type, at the lower right in the Status Bar. This is a good place to look to make sure you have verified a new or modified study before attempting to place it on a chart. You'll also notice that the Status Bar now contains the word SAVED in dark type. Documents are saved whenever you attempt to verify, even when there are errors.

In addition, you will see the following message from the Development Environment:



This message reminds you that when modifying and then verifying an EasyLanguage study that is currently applied to a chart (or RadarScreen or OptionStation window); changes to that study are automatically recalculated and applied to that chart. There is a checkbox option you can select if you don't wish to see this message again.



If any of your grammar or syntax is incorrect, the errors will be indicated in the *Output* tab of the Development Environment. Correct any errors and then verify again to save the changes.

Tip: The Output bar can be set to Auto Hide. If so, you may click on the Output tab to view the Output bar, and you may want to 'pin' the Output bar to the Development Environment so that it stays visible. You may do this using the View – Toolbars – Output menu sequence, or the controls visible in the upper right of the Output bar.

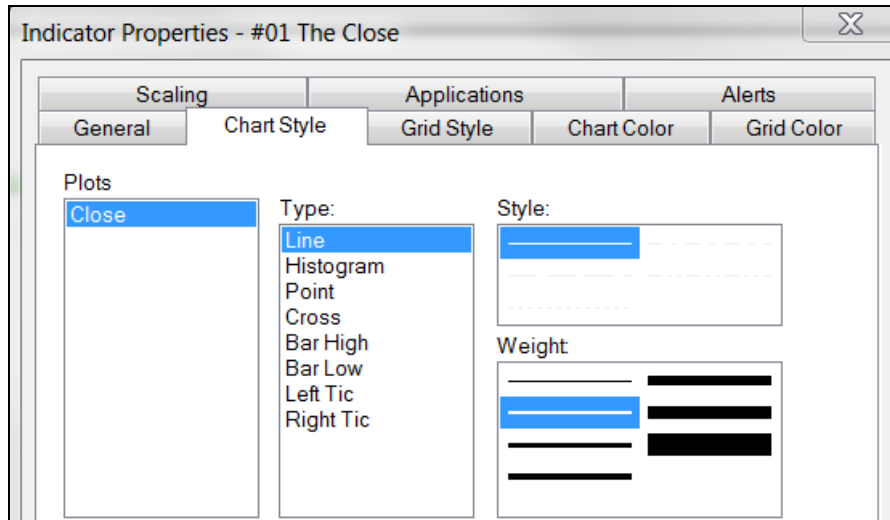
Tip: It is possible to save EasyLanguage documents that are “works in progress” without verifying them. If you are working on a document which you know is not going to verify because it is incomplete, select *File* then *Save* or *Save As* from the main menu in the Development Environment. The study is then saved in its current state and can be completed at a later time.

Before placing the **#01 The Close** indicator on a chart, you should become familiar with and learn how to modify properties settings for EasyLanguage documents.

7. Setting Properties for EasyLanguage Documents

When creating EasyLanguage studies, you have the option of setting defaults for items such as color, style, and alert specifications. In TradeStation, this is referred to as modifying or setting the **properties** for analysis techniques and strategies.

A right-click in the Editor window for *#01 The Close* will bring up a shortcut menu, and selecting *Properties* at the bottom of the menu will bring up the *Indicator Properties* dialog box.



This dialog contains eight tabs that are used to view and change the indicator's formatting options such as color, style, scaling, and alerts. Settings made here will be saved and used each time this indicator, *#01 The Close*, is applied to a Chart Analysis or RadarScreen (or OptionStation) window. Try making some modifications to the properties settings. In the *Chart Style* tab, change the weight of the line to make it a bit thicker. In the *Chart Color* tab, change the color of the plot from red to blue and in the *Scaling* tab, change the *Scale On* setting from *Right Axis* to *Same Axis as Underlying Data*. Now, click *OK* and re-verify the indicator.

Note: The default scaling for indicators is *Right Axis*, which places the study in the first available sub-graph on the chart; in other words, beneath the current underlying data, using the right axis as a scaling reference. For this exercise, it makes sense to overlay this indicator on the price bars, which is why *Same Axis as Underlying Data* should be selected.

Important note regarding verification of EasyLanguage studies: When creating EasyLanguage documents remember the following:

Verify → Properties → Verify

After creating and verifying an analysis technique, you may then set the properties in the appropriate dialog. After any modifications are made and the dialog is exited, you must then **re-verify** the analysis technique. If you fail to re-verify after modifying the properties settings the analysis technique will plot, however, any modifications made to the properties settings will not be evident.

Now return to the TradeStation Platform, create a chart and insert the indicator. **#01 The Close** plots the closing prices as a blue line which overlays the price bars. However, after seeing the indicator on the chart you may decide to change the plot color for **#01 The Close** to red. In addition to modifying properties for EasyLanguage studies from the Editor, properties settings may also be accessed and modified from a Chart Analysis, RadarScreen or OptionStation window. To do this, right click on the chart and select *Format Analysis Techniques*. Click the *Format* button in the dialog that appears and on the *Color* tab change the color from blue to red. This lets TradeStation know that although this indicator has a default color of blue you'd like to plot it as a red line in this particular chart. To instruct TradeStation to plot **#01 The Close** as a red line in all charts in the future, in other words to make this change the new default, you would click the *Default* button. Do not do this at this time.

Click *OK* in the *Format Indicator* dialog and click *Close* in the *Format Analysis Techniques and Strategies* dialog to return to the chart, which now contains the indicator plotted with a red line.

Tip: Each type of analysis technique in TradeStation (indicator, ShowMe, PaintBar, etc.) has its own properties dialog with specific formatting options. Strategies also have a properties dialog. These settings may be accessed when working in the Editor of the Development Environment or from the appropriate window in the TradeStation Platform when using them.

***01 The Close**



Exercise: *02 Volume

Learning objective: Using plot statements in indicators.

Description: This indicator plots the volume (shares or contracts) or the tick count (transactions), depending on the type of chart and the formatting settings.

In the Development Environment, use the *File – New – Indicator* menu sequence, or the New toolbar button, to create a new indicator named **#02 Volume**. In the Editor window, type the following:

```
Plot1(Ticks, "Volume");
```

Note: The reserved word `Ticks`, under most circumstances, will return the desired trade volume or tick count based on your settings in the *Format Symbol* dialog. Depending on the symbol and chart type you are using, other volume-related reserved words might be more appropriate. This information is covered in Appendix A in a section titled Volume and Ticks.

Verify the indicator and then go to the *Indicator Properties* dialog (right click in the document window and select *Properties* from the shortcut menu to do this).

Click the *Scaling* tab. You will notice that the default setting for *Scale On*, which is *Right Axis*, is selected. This will result in this indicator being plotted in its own sub graph, unlike **#01 The Close**, which was overlaid on the price bars as a result of selecting the *Scale On* setting, *Same Axis as Underlying Data*. Since you would like this indicator to plot in its own sub graph, leave the scaling set to *Right Axis*. Also take a look at the *Chart Style* tab. You'll notice that the default setting for *Type* is *Line*. Change this setting to *Histogram* and, if you like, make the weight of the line a little thicker. Notice also the word `Volume` in the *Plots* box. This is the *plot name* "Volume" you specified when writing this indicator.

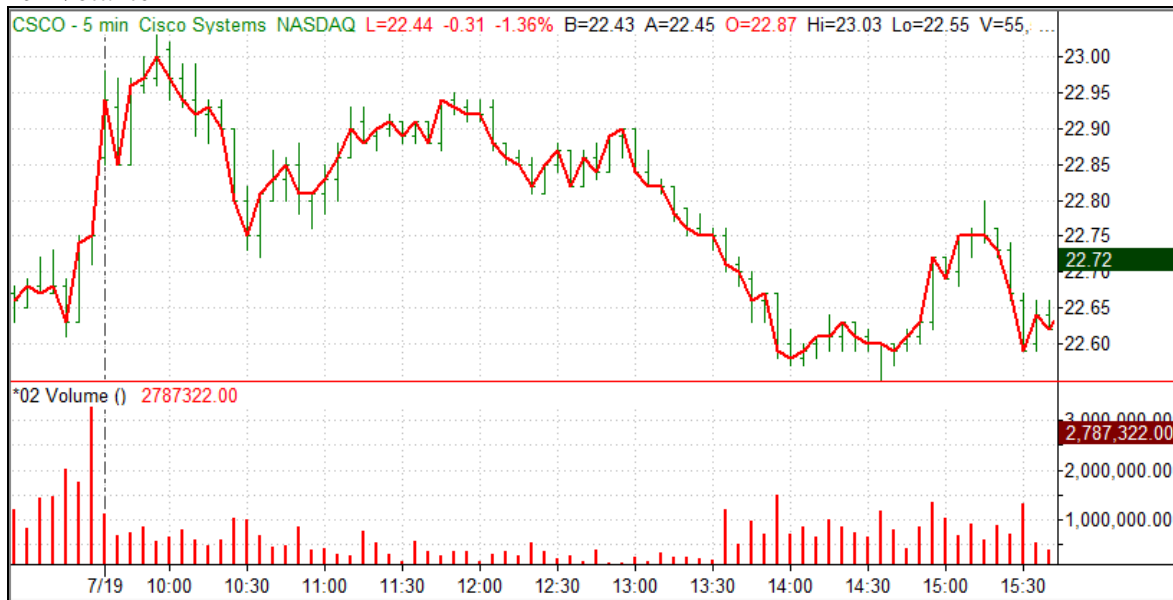
```
Plot1(Ticks, "Volume");
```

If this indicator contained multiple plot statements, each different plot name would be listed in the *Plots* box. Again, although adding plot names is optional when writing plot statements, you can see how this can be a useful reference, especially when there are multiple plots in an analysis technique.

Click *OK* to exit this dialog. Remember to verify the indicator again in order to save the properties settings which were just modified.

Return to the TradeStation Platform, to the previously created Chart Analysis window and insert the newly created **#02 Volume** Indicator. It should look something like the figure below, which contains both **#01 The Close** and **#02 Volume**.

*02 Volume



You're now ready to attempt your first Challenge. Remember to attempt each Challenge on your own.

Challenge 1: *03 High and Low

Learning objective: Using plot statements in indicators.

Write an indicator that plots a line connecting the highs of each bar and a second line connecting the lows of each bar.

Set the properties to:

- Display the plots as lines, and

- Display the plots in the same sub-graph as the price data.

Hint: This indicator will require 2 plot statements. The reserved word in EasyLanguage for the high is `High`. The reserved word in EasyLanguage for the low is `Low`.

Tip: If your Chart Analysis window starts to appear too cluttered from applying all the analysis techniques you have inserted, you can remove some of them from the chart. Remember, you will not lose them as they have all been verified and saved.

Note: Answers for each Challenge may be found in the Appendix of this book and as a recorded video piece with audio narration at: <http://www.tradestation.com/ELHSC>

8. Mathematical Operators

EasyLanguage can perform addition, subtraction, multiplication, and division. These mathematical operators may be used to perform calculations within analysis techniques and strategies:

Addition	+
Subtraction	-
Multiplication	*
Division	/

The next exercise, named *Real Body*, focuses on using a mathematical operator in the numeric expression in a plot statement. If you're familiar with Candlestick charts, you'll be familiar with the importance of understanding the relationship between the open and close. That relationship, which in Candlestick charts is graphically displayed as a rectangle, is known as the "real body," as shown below. This exercise illustrates how EasyLanguage may be used to convert analyses commonly done visually, like Candlestick analysis, to mathematical formulas.



Exercise: *04 Real Body

Learning objective: Using mathematical operators in the numeric expression in a plot statement.

Description: This indicator plots the difference between the `Close` and the `Open` of each bar. This will be a positive value if the `Close` is greater than the `Open` and a negative value if the `Close` is less than the `Open`. This corresponds to the "real body," green or red, of a Candlestick.

In the Development Environment, create a new indicator named **#04 Real Body**. Type the following plot statement:

```
Plot1(Close - Open, "RealBody");
```

With a bullish candle (if the `Close` is greater than the `Open`), the formula will return a positive value to be plotted. With a bearish candle (if the `Close` is less than the `Open`), the result will be a negative. Therefore this indicator will have an oscillator quality to it. For that reason, it might be a good idea to include a "zero-line" as a reference. Add this additional plot statement (`Plot2`) and name the plot "*Doji*."

```
Plot1(Close - Open, "RealBody");  
Plot2(0, "Doji");
```

Verify the indicator and then go to the *Indicator Properties* dialog. (Right click in the Editor window and select *Properties* from the shortcut menu to do this).

Take a look at the *Scaling* tab and ask yourself where this indicator should appear on the chart. Since it is an oscillator, the default *Scaling* setting, *Right Axis*, should be fine, placing this indicator in its own sub-graph below the price data.

Click the *Chart Color* tab to change the color of either of the plots. Notice the plot names `RealBody` and `Doji` in the *Plots* box on this tab. These are the plot names we used when writing the indicator. Click *OK* and remember to re-verify the indicator. Now return to the TradeStation Platform and insert **#04 Real Body** into a chart. It should look like the figure below.

***04 Real Body**



9. Order of Operations

When writing EasyLanguage studies, you may use multiple mathematical operations within a single numeric expression. Therefore, in order to achieve the desired result, it is critical that you understand and can manage the order of operations. In a mathematical expression that includes addition, subtraction, multiplication, and division, TradeStation calculates arithmetic operations as follows:

Multiplication and division, from left to right

followed by:

Addition and subtraction, from left to right

This order may be customized by use of parentheses. Operations inside parentheses are done first, starting with the innermost parentheses and working outward:

$$1 + 5 * 3 + 4 = 20$$

$$(1 + 5) * (3 + 4) = 42$$

$$1 + (5 * (3 + 4)) = 36$$

As you can see, the use of parentheses to determine the order of operations will produce very different results in expressions that contain the same numbers and operators.

Let's look at two exercises which illustrate the concept of order of operations.

Exercise: *05 MidPrice

Learning objective: Using parentheses in numeric expressions to order operations correctly.

Description: This indicator plots a line connecting the middle price of each bar.

What mathematical formula may be used to find the middle price for each bar? Let's look at an example.

If the high of a bar is 10 and the low is 4, what is the mid-price? In English, that would be 10 plus 4 divided by 2, which would return a value of 7. In EasyLanguage we could write:

$$\text{High} + \text{Low} / 2$$

Would that result in the correct value? Let's see. This numeric expression contains two mathematical operations, **addition** and **division**. Remember, although EasyLanguage works from left to right, multiplication and division operations are always done before addition and subtraction.

Following these rules, the output in EasyLanguage would be 12, which is not correct, because the Low, in this case 4, is **first** divided by 2 and **then** added to the High:

$$\begin{array}{r} 10 + 4 / 2 = \\ 10 + 2 = \\ 12 \end{array}$$

To return the correct value, the High and Low must **first** be added together with the sum **then** being divided by 2. We'll need to employ a set of parentheses as follows to achieve the correct result:

$$\begin{array}{r} (\text{High} + \text{Low}) / 2 \\ (10 + 4) / 2 \\ 14 / 2 = 7 \end{array}$$

Surrounding the addition operation with parentheses achieves the desired result. The High and Low are added together first, with the sum (14) then being divided by 2.

Note: Whenever possible, defer to multiplication instead of division when the result is exactly equivalent. This is recommended because TradeStation does a check for "divide by zero errors" that may cause division to be performed a bit more slowly than multiplication. In the above example, we would use **"*.5"** instead of **"/ 2"** since the result is the same. However, using **"/ 3"** is generally preferable to **"*.33"** for reasons of accuracy, speed notwithstanding.

Now, let's write this indicator.

In the Development Environment, create a new indicator named **#05 MidPrice**. Type the following plot statement:

```
Plot1((High + Low) * .5, "MidPrice");
```

Verify the indicator and access the *Scaling* tab of the *Properties* dialog. Change the scaling from *Right Axis* to *Same Axis as Underlying Data* so this indicator overlays the price bars. Click *OK* and re-verify to save the change.

You may now insert this indicator to a chart. It should resemble the figure below.

***05 MidPrice**



Exercise: *06 VolWtdRange

Learning objective: Using parentheses in numeric expressions to order operations correctly.

Description: This Indicator calculates the range of each bar multiplied by its volume, resulting in a volume-weighted range indicator.

In the Development Environment, use the *File – New –Indicator* menu sequence, or the New document toolbar button, to create a new indicator named **#06 VolWtdRange**. Type the following plot statement:

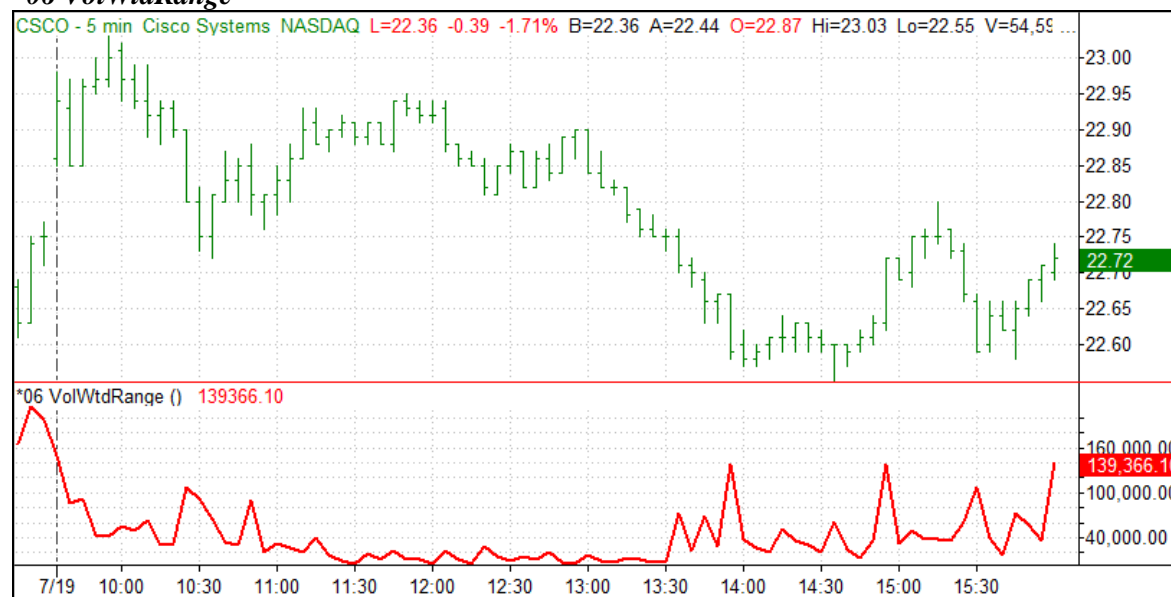
```
Plot1( (High - Low) * Ticks, "VWR" );
```

As in the previous exercise, parentheses are used to achieve the correct order of operations. In this case, the desired value is the bar's range multiplied by its volume. Placing *High* and *Low* in parentheses correctly subtracts one from the other **before** multiplying by the volume.

Verify the indicator, then right-click in the Editor window and select *Properties* from the shortcut menu if you want to change any of the default properties. It is not necessary to change the Scaling setting since the default *Scale On: Right Axis* is appropriate for this indicator.

You may now insert this indicator to a chart. It should resemble the figure below.

*06 VolWtdRange



10. Referencing Data from Previous Bars

Previously occurring prices and other values may be referenced using square brackets, [], immediately following the name of the value to be referenced.

For example:

The close of the bar 5 bars ago would be written:

`Close[5]`

The Range of the bar 12 bars ago would be written:

`High[12] - Low[12]`

or

`(High - Low)[12]`

but not

`High - Low[12]`

which would subtract the low of 12 bars ago from the current bar's High.

Now let's try a few exercises to illustrate this EasyLanguage concept.

Exercise: *07 NetChangeOsc

Learning objectives: Referencing data from previous bars.

Description: This indicator plots a 5-bar net change oscillator.

Create a new indicator named **#07 NetChangeOsc**. In the Editor, type the following plot statement:

```
Plot1(Close - Close[5], "NCO");
```

The `Plot1` statement you've written above subtracts the close of 5 bars ago from the current `Close` and assigns the plot name `NCO`.

Since this indicator will have an oscillator quality to it, it might be a good idea to include a “zero-reference line.” Add this additional plot statement (`Plot2`) and name the plot `0`.

```
Plot1(Close - Close[5], "NCO");  
Plot2(0, "0");
```

Verify the study. Since the default scaling is *Right Axis*, which places this indicator in its own sub-graph, there are no properties settings that need modifying. Insert this indicator into a chart. It should resemble the figure below.

****07 NetChangeOsc***



Challenge 2: *08 Bands

Learning objective: Using square brackets to reference previous values.

Write an indicator that plots two bands around the price data.

One band adds half the previous bar's range to the current bar's opening.

The other band subtracts half the previous bar's range from the current bar's opening.

Remember: The range of a bar is high minus low. How would you express the range of the previous bar?

This indicator should overlay the price bars; therefore the *Scale On* setting on the *Scaling* tab in the *Indicator Properties* should be set to *Same Axis as Underlying Data*.

11. Numeric Variables

A numeric variable is a value or formula given a name for easy reference. The name may then be used to refer to that value or formula repeatedly within a single EasyLanguage document without having to re-type the formula for that value. Up to this point, we have been writing our formulas directly in our plot statements. However, EasyLanguage allows us to remove the formula from the plot statement, give it a name, and then reference the given name in the plot statement. This will be especially useful when creating EasyLanguage studies that use multiple and complex formulas many times within the analysis technique or strategy being created. Using variables can improve the efficiency of EasyLanguage because a formula assigned to a variable needs to be calculated only once for each bar and may then be referenced by the variable name.

Numeric variables are used to:

- Improve processing efficiency
- Reduce typographical errors
- Increase readability

Note: A variable is only valid within the EasyLanguage document in which it is declared and/or assigned.

There are two types of numeric variables: pre-declared numeric variables and user-declared numeric variables.

User-declared Numeric Variables

EasyLanguage allows for creation of custom names for numeric variables. That is, you may “declare” the names you would like to use for variables; EasyLanguage allows you to add words, your words, to the existing language. The benefit is that since you create the names for these variables, they may be descriptive of the value assigned to that variable. To do this, you must first declare the names as numeric variables and then a numeric expression must be assigned to the name. This requires two steps, with two new statements you have not yet seen: a variable declaration statement and a variable assignment statement.

Variable Declaration Statement

This statement declares a numeric variable named `NetChange` with an initial value of 0:

```
Variables: NetChange(0);
```

Variable Assignment Statement

This statement assigns the numeric expression for net change to the variable:

```
NetChange = Close - Close[1];
```

Now let's try an exercise that uses user-declared numeric variables.

Exercise: *09 Bands2

Learning objective: Using user-declared numeric variables; writing variable declaration statements and variable assignment statements.

Description: This is a re-write of Challenge #2, *08 Bands, incorporating user-declared numeric variables.

Create a new indicator named *#09 Bands2*. If you recall, the EasyLanguage for the original indicator, *#08 Bands* looked like this:

```
Plot1(Open + (High[1] - Low[1]) *.5, "HighBand");  
Plot2(Open - (High[1] - Low[1]) *.5, "LowBand");
```

This indicator plotted two bands around the price data. One plot added half the previous bar's range to the open and the other subtracted half the previous bar's range from the open.

Question: What is identical in both the above plot statements?

Answer: The formula for calculating half the previous bar's range is the same in both plot statements:

```
(High[1] - Low[1]) * .5
```

Therefore, to make this more efficient, the formula can be removed from the plot statement and assigned to a user-declared variable. To accomplish this, first type the following variable declaration statement in the document window. We will add a few other variables as well. Note that you may declare multiple variables in a single declaration statement and may use `Var:` or `Vars:` or `Variable:` or `Variables:` to begin the declaration statement.

```
Vars: HalfPrevRange(0), HiBand(0), LoBand(0);
```

This statement declares three numeric variables and sets their initial values to zero. Next, type the following variable assignment statements, which assign numeric expressions (formulas) to each of the variables we declared in the declaration statement above.

```
Vars: HalfPrevRange(0), HiBand(0), LoBand(0);
```

```
HalfPrevRange = (High[1] - Low[1]) * .5;  
HiBand = Open + HalfPrevRange;  
LoBand = Open - HalfPrevRange;
```

Notice that one of the variables, `HalfPrevRange`, which calculates half the previous bar's range, is used in the other two variables, `HiBand` and `LoBand`, which calculate the open plus half the previous bar's range, and the open minus half the previous bar's range, respectively.

Now you may write the two plot statements as they are printed below:

```
Vars: HalfPrevRange(0), HiBand(0), LoBand(0);
```

```

HalfPrevRange = (High[1] - Low[1]) * .5;
HiBand = Open + HalfPrevRange;
LoBand = Open - HalfPrevRange;

```

```

Plot1 (HiBand, "HighBand");
Plot2 (LoBand, "LowBand");

```

Plot1 plots the user-declared numeric variable HiBand, which adds half the previous bar's range to the open. Plot2 plots the user-declared numeric variable LoBand, which subtracts half the previous bar's range from the open.

Verify the indicator. The resulting plot from this indicator should look identical to the plot of the earlier completed **Challenge #2 *08 Bands**.

*09 Bands2



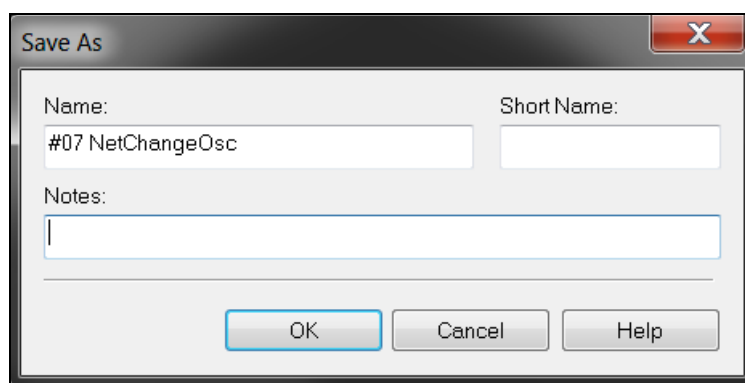
Exercise: *10 NetChangeOsc2

Learning objective: Using user-declared numeric variables; writing variable declaration statements and variable assignment statements.

Description: This is a re-write of *07 NetChangeOsc, incorporating user-declared numeric variables.

In the Development Environment, click the tab or Editor window for *#07 Net ChangeOsc*. If it is not already open, use the *File – Open* menu sequence to open this indicator.

In this exercise, you'll rewrite the previous version using a user-declared variable for the calculation of the net change oscillator. To save the previous exercise with the new name, make sure your *#07 Net ChangeOsc* is active and select *File - Save As...* from the main menu. You'll see the following dialog asking you to name the file:



Rename the file, *#10 NetChangeOsc2* and click *OK*. TradeStation creates a copy of the original indicator with the new name, while still saving the original.

The prior exercise looked like this:

```
Plot1(Close - Close[5], "NCO");  
Plot2(0, "0");
```

The formula for net change oscillator, $\text{Close} - \text{Close}[5]$, will be assigned to a variable. This requires 2 steps: first you must declare the name that you will use for the variable and its initial value:

```
Vars: NCO(0);
```

Then assign the formula to the variable and use the name of the variable in the plot statement:

```
NCO = Close - Close[5];  
  
Plot1(NCO, "NCO");  
Plot2(0, "0");
```

Verify the indicator. The resulting plot from this indicator should look identical to the plot of the earlier completed #07 *NetChangeOsc*.

*10 NetChangeOsc2



Pre-declared Numeric Variables

Pre-declared numeric variables have names which already exist in the EasyLanguage vocabulary. All you need to do is tell TradeStation what the name means, in other words, assign a value or numeric expression to the name.

One hundred names have been reserved for numeric variables. These names are:

Value0 through Value99.

Value0 through Value99 are automatically recognized as numeric variables and only require the assignment of a numeric expression to the name. This is accomplished by using a **variable assignment** statement:

Value0 = *numeric expression*;

As an example, to use the name Value1 to refer to the net change throughout an analysis technique or strategy you would type:

Value1 = Close - Close[1];

Pre-declared variables offer convenience but lack the readability of user-declared variables. This is particularly noticeable in long documents with many variables.

12. Using Functions

Functions are frequently used formulas (or comparisons) written in EasyLanguage that return numeric values (or string/text or true/false). They may be called for use in any analysis technique or strategy with just a few words, eliminating the need to re-create or repeatedly re-type complex formulas.

- Functions can be used in any set of EasyLanguage rules or instructions.
- Most common mathematical and statistical formulas such as average and standard deviation are already stored as functions in EasyLanguage.
- Also, most common technical analysis formulas are stored as functions, such as RSI and ADX.
- Many functions require the user to supply parameters for the function.
- Functions may also be created by the user.

Fortunately, the EasyLanguage Dictionary includes hundreds of functions, in addition to Reserved Words, in its listings. This will save an enormous amount of time when writing EasyLanguage documents. For example, let's say you wanted to calculate a 20 bar moving average. You could type the following to accomplish this:

```
Value1 = (Close + Close[1] + Close[2] + Close[3] + Close [4] +  
Close[5] + Close[6] + Close[7] + Close[8] + Close [9] +  
Close[10] + Close[11] + Close[12] + Close[13] + Close [14] +  
Close[15] + Close[16] + Close[17] + Close[18] + Close  
[19])/20;
```

At this point you're probably wondering if there is an easier way. There is. Typing the words that call the `Average` function or dragging and dropping them into the Editor from the Dictionary, would accomplish the same task much more quickly and efficiently. In fact, many useful mathematical and technical analysis functions and reserved words may be found in the EasyLanguage Dictionary. Here you will find prewritten functions for RSI, ADX, DMI, Stochastic and many more.

An additional benefit to using the EasyLanguage Dictionary is that the Description pane provides information on the parameters and the values returned. For functions and reserved words that require parameters, the name is followed by these parameters in parentheses. So, the function for `Average` that you would use to calculate a moving average, for example, contains parameters for both `price` and `length`:

```
Average (Price, Length)
```

When using functions it is necessary for the user to provide values for these parameters. So, if we wanted to use the `Average` function based on the closing price and a `Length`, or number of bars of **14**, it would look like this:

```
Average (Close, 14)
```

Functions may be dragged and dropped directly from the EasyLanguage Dictionary into an EasyLanguage document. We'll examine how to do this in the following two exercises.

Exercise: *11 Momentum

Learning objective: Calling a function into an EasyLanguage document; using a user-declared numeric variable.


Description: This indicator plots the 10-bar momentum of closing prices.

Create a new indicator named *#11 Momentum*. In this exercise, create a user-declared numeric variable named *Mom* and initialize it to zero. In addition, write the first part of the variable assignment statement:

```
Vars: Mom (0);
```

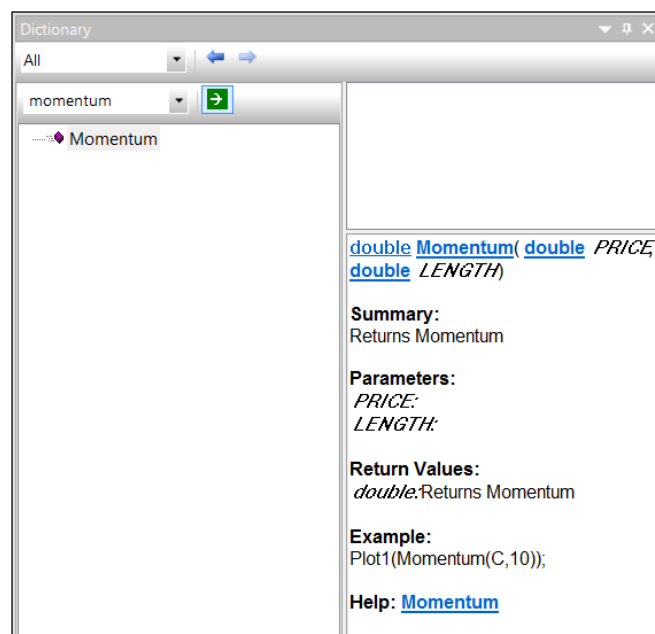
```
Mom =
```

Before we use the EasyLanguage Dictionary to find a function that will calculate momentum for us, let's have a more detailed look at the Dictionary. Roll the mouse pointer over (or click on) the Dictionary tab at the right of the Development Environment. You may also use the *Tools* menu

or the Dictionary button  on the Toolbar. As described at the beginning of the course, you will see the Objects pane in the left-hand section of the Dictionary. Click the '+' sign next to *easylanguage.ell*. This is the sub-section of the Dictionary that contains our entries of interest for this course. You will see the list of categories for functions and Reserved Words. Click on any category and you will see its members in the Members pane to the right. For example, click on Date and Time and you will see items such as *CurrentTime*, *Dayofweek* and *Time*. Now, click on one of the items in the Members pane, such as *Date*, and you will see information about that item in the Description pane below.

Now we will use the Search feature to find the momentum function we need for this exercise. Type 'momentum' in the Search field above the Objects pane, then click on the arrow in the green box to the right. In the Objects pane, the Momentum function should appear. Click on the function and the Description pane will display relevant information, including the parameters that must be supplied.

Drag the Momentum function from the Objects pane and drop it into the Variable Assignment statement in the Editor.



As mentioned earlier you will now need to provide the parameters for the *Momentum* function, in parentheses following the name of the function. Note that the left parenthesis was supplied when you dropped the function in from the Dictionary, and notice also that the tool tip appears reminding you of the parameters Price and Length. Complete the variable assignment statement as it appears below. For this indicator, the calculation is the 10-bar momentum of closing prices.

```
Vars: Mom (0);
```

```
Mom = Momentum(Close, 10);
```

Now add the following plot statements to plot the *Momentum* indicator and a reference or zero line:

```
Vars: Mom (0);
```

```
Mom = Momentum (Close, 10);
```

```
Plot1(Mom, "Momentum");
```

```
Plot2(0, "zero");
```

After verification, plot this indicator to view something similar to the *Momentum* indicator plotted in the figure below.

*11 Momentum



Exercise: *12 Real Body Avg

Learning objective: Calling a function into an EasyLanguage document; using a user-declared numeric variable.

Description: This indicator plots the 10-bar average of the difference between the close and the open of each bar, as originally plotted in **04 Real Body*.

In the Development Environment, create a new indicator named *#12 Real Body Avg*. Declare two variables: RB for real body and AvgRB for the real body average:

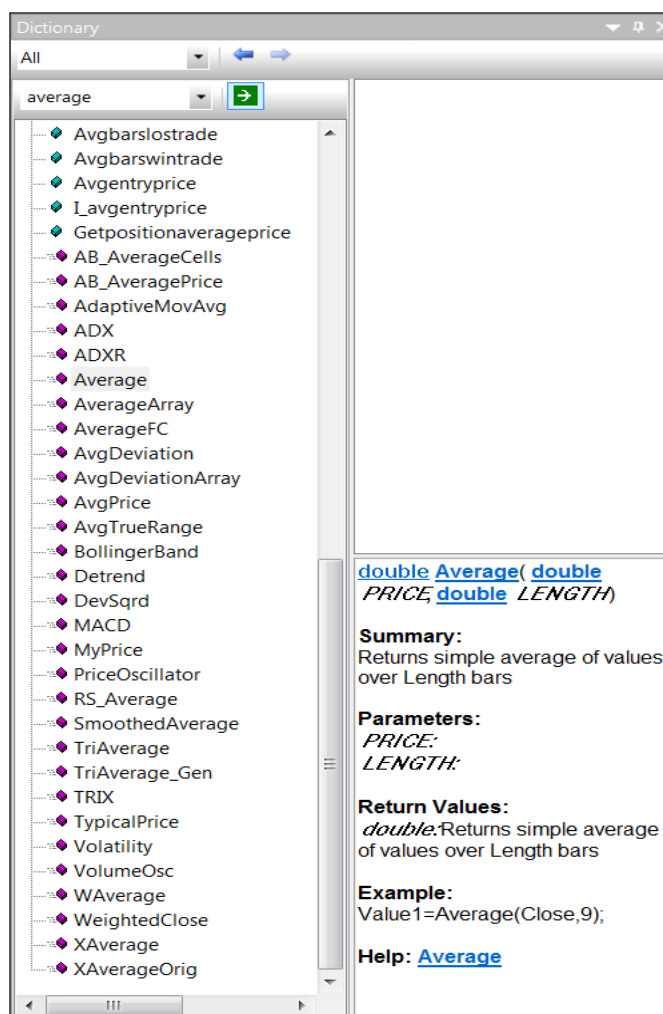
```
Vars: RB(0), AvgRB(0);
```

The first assignment statement will calculate the Real Body or Close - Open:

```
Vars: RB(0), AvgRB(0);
```

```
RB = Close - Open;
```

The second assignment statement will use the Average function, which can be found in the EasyLanguage Dictionary, to calculate an average real body value over a specified length. Immediately after typing the first part of the 2nd variable assignment statement, AvgRB =, go to the EasyLanguage Dictionary and use the *Search* feature to find the *Average* function. To do this, type the word Average in the *Search* field at the top of the dictionary, and then click on the arrow to the right. Scroll down through the list displayed in the Members pane until you find Average.



Select *Average* and notice the parameters as displayed in the Description pane. Drag the function to the Editor window and drop it into the AvgRB variable assignment statement.

Note the tip prompting you for the required parameters and complete the statement as below:

```
Vars: RB(0), AvgRB (0);  
  
RB = Close - Open;  
AvgRB = Average (RB, 10);
```

It's important to note that the parameter for *Price* in the *Average* function is really a placeholder for any numeric value found on the bars. In this exercise we are using RB, which was set to the numeric value *Close - Open* in the variable assignment statement.

Now add two plot statements, one for the real body average and one for a zero reference line. The finished document should look like this:

```
Vars: RB(0), AvgRB(0);  
  
RB = Close - Open;  
AvgRB = Average(RB, 10);  
  
Plot1(AvgRB, "AvgRB");  
Plot2(0, "Doji");
```

There's no need to adjust the properties for *Scaling* since this indicator will default to plot in its own sub-graph. Verify the indicator and insert it into a chart.

***12 Real Body Avg**



Challenge 3: *13 Envelope

Learning objective: Calling a function into an EasyLanguage document; using user-declared numeric variables.

Write an indicator that plots a 20-bar moving average of the highs and a 20-bar moving average of the lows.

Select names and declare variables for the two moving average values. This will require variable assignment statements, too.

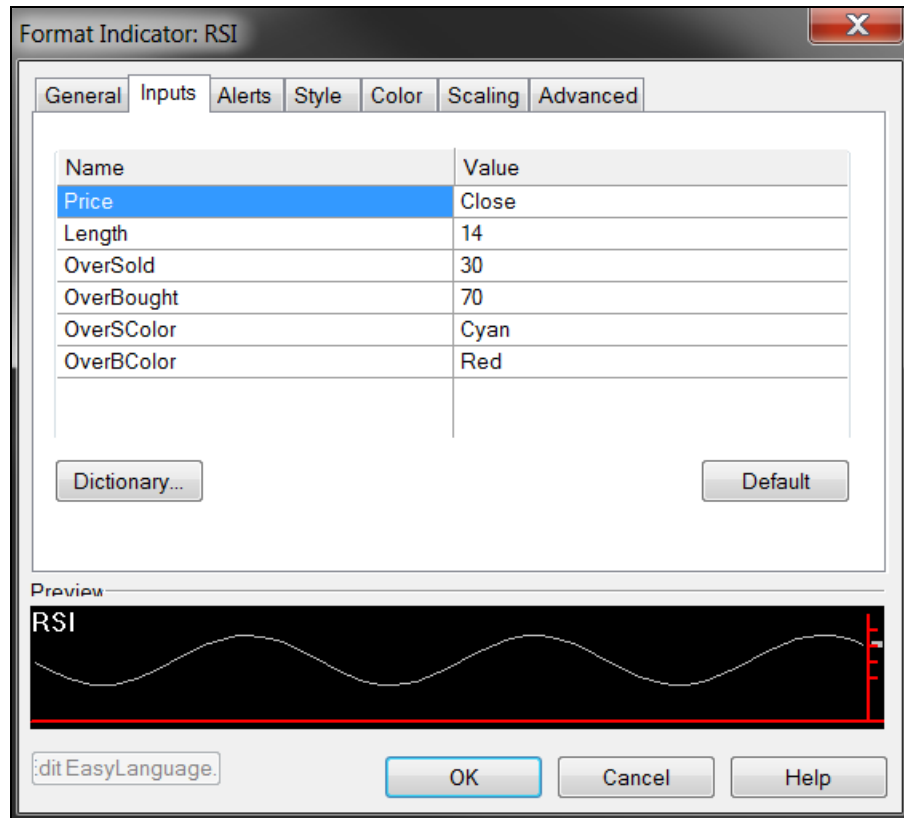
This indicator should plot in the same sub-graph as the price data.

Hint: The `Average` function may be dragged in from the Dictionary.

13. Inputs

An input is a user-editable value used in an analysis technique or strategy. It is a place holder which allows the user to change a value from outside the EasyLanguage Editor; specifically, from the *Inputs* tab of the *Format* dialog. This provides flexibility and efficiency when modifying analysis techniques and strategies.

The figure below shows the *Input* tab in the *Format Indicator* dialog for TradeStation's *RSI* indicator. As you can see, six inputs are specified and can be changed directly from this dialog:



So, if you decided to change the *Length* used in the *RSI* calculation, or the numeric levels indicating *OverBought* and *OverSold* conditions, you could easily modify the indicator without having to return to the Editor in the Development Environment. To set any changes as the new defaults, click the *Default* button before clicking *OK*.

Tip: Note that the *Value* fields on the *Inputs* tab above contain straightforward numeric expressions (the words for colors represent numeric values). However, numeric inputs may contain formulas and even functions; true/false inputs may contain true/false expressions or functions that return true or false.

The benefits of inputs:

- Each input carries a user-declared name and default value.
- Input values may be edited while formatting without returning to the Development Environment.
- Inputs are mandatory for strategy optimization.
- Inputs allow for multiple instances of an analysis technique or strategy each using different calculation values.

Inputs are declared using an input declaration statement and are assigned a default value within the statement:

```
Input: InputName(default value);
```

The following declares an input named `Factor` with a default value of `1.005`:

```
Input: Factor(1.005);
```

Note: Each input is only valid within the analysis technique or strategy in which it is declared.

Exercise: *14 Envelope2

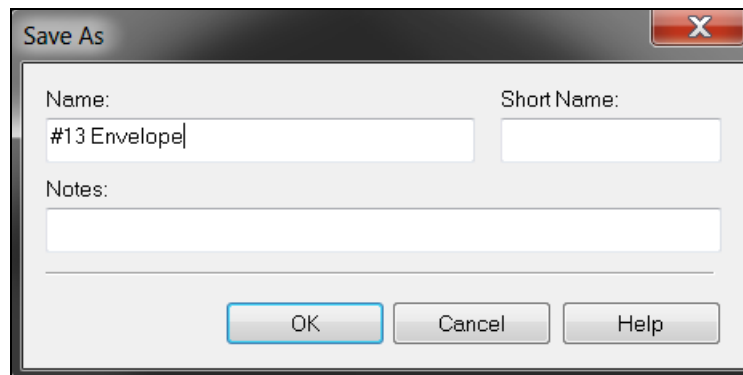
Learning objective: Using inputs in place of fixed values.

Description: This is a re-write of Challenge 3, **13 Envelope*, incorporating inputs for the lengths of the moving averages.

In this exercise, you will rewrite the previous challenge using inputs for the lengths of the moving averages.

In the Development Environment, access the tab or Editor window for **Challenge 3, #13 Envelope**. If it is not open, use the *File – Open* menu sequence to open this indicator.

To save the challenge with a different name, make sure your **#13 Envelope** is active and select *File - Save As...* from the main menu. You'll see the following dialog asking you to name the file:



Rename the file **#14 Envelope2** and click *OK*. TradeStation creates a copy of the original indicator with the new name, while still saving the original.

To complete this exercise, change the specified length in the variable assignment statements to an input. Thus, the two existing variable assignment statements using a specified length of 20 in the Average function change from:

```
UpperLine = Average(High, 20);  
LowerLine = Average(Low, 20);
```

to:

```
UpperLine = Average(High, UpperLength);  
LowerLine = Average(Low, LowerLength);
```

Of course, these inputs need to be declared and this should be done with the input declaration statement:

```
Input: UpperLength(20), LowerLength(20);
```

The finished product should look like this:

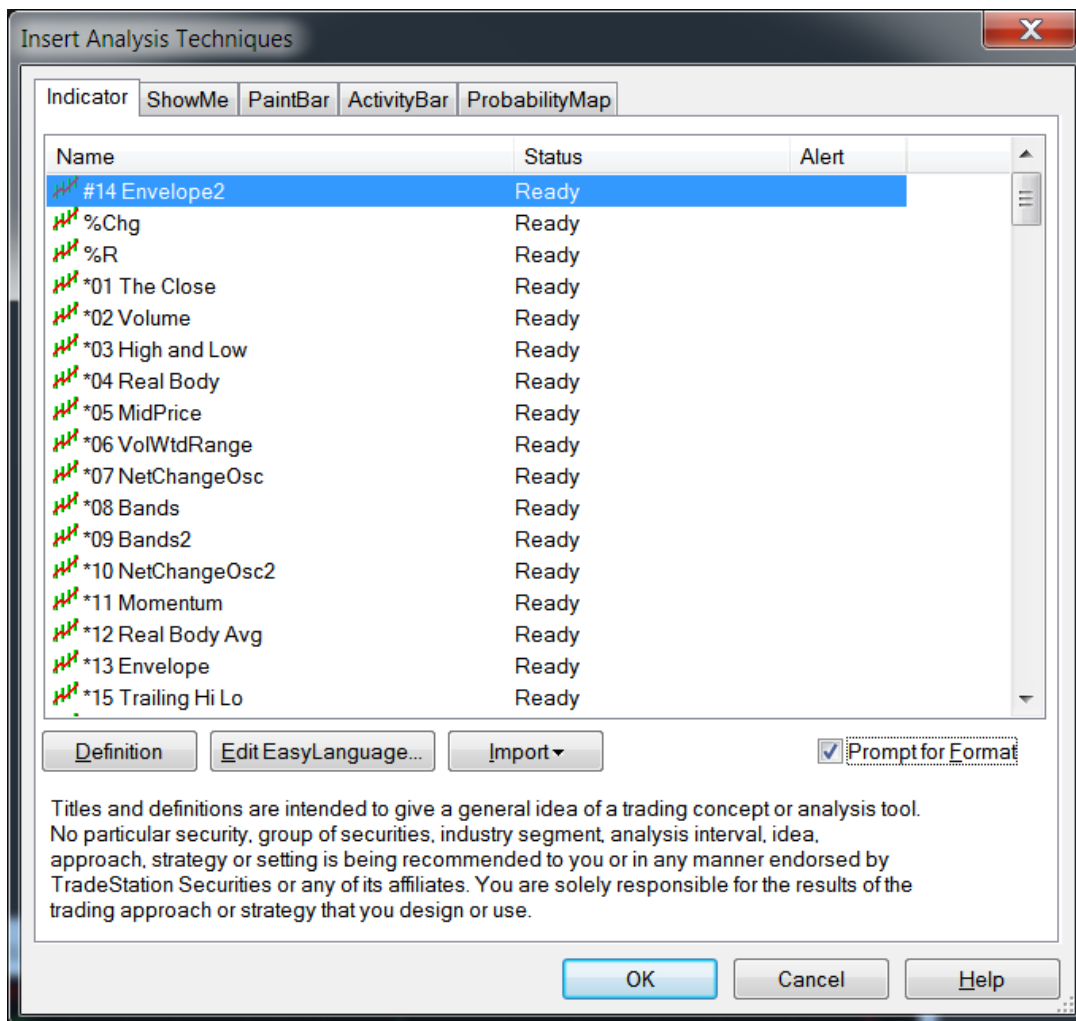
```
Input: UpperLength(20), LowerLength(20);

Vars: UpperLine(0), LowerLine(0);

UpperLine = Average(High, UpperLength);
LowerLine = Average(Low, LowerLength);

Plot1(UpperLine, "UpperLine");
Plot2(LowerLine, "LowerLine");
```

Verify the indicator. When inserted in a chart, it will look identical to **Challenge 3**. Go to a Chart Analysis window and insert **#14 Envelope2**; however, before clicking **OK**, make sure the **Prompt for Format** checkbox is checked as in the figure below:



Click **OK** and select the **Inputs** tab. You can see the specified inputs, **UpperLength** and **LowerLength**, which you created. Clicking **OK** will place this on the chart.

*14 Envelope2



Rule of thumb for sequence of statements:

As you can see from all the exercises you've completed, the sequence of statements has followed a specified order in each EasyLanguage document. As a rule-of-thumb for now, let's say the following order should be applied:

Input declaration statement

Variable declaration statement

Variable assignment statement(s)

For analysis techniques: Plot statement(s)

For strategies: Buy and sell statement(s) (to be covered later)

However, when creating EasyLanguage documents, especially the more extensive and complex variety, it is very often easier to work from the bottom-up, laying out the framework of the document, and then returning to fill in the blanks. Although, according to our rule-of-thumb, these statements need to end up in the above specified order to properly verify and execute, they do not need to be crafted in this order initially.

For example, if you know you need two plot statements in an indicator, you could write them first and assign them plot names:

```
Plot1(UpperLine, "UpperLine");  
Plot2(LowerLine, "LowerLine");
```

Then you'd have to assign values to each of the two variables:

```
UpperLine = Average(High, UpperLength);  
LowerLine = Average(Low, LowerLength);
```

```
Plot1(UpperLine, "UpperLine");  
Plot2(LowerLine, "LowerLine");
```

Declare the variables that were just assigned:

```
Vars: UpperLine(0), LowerLine(0);
```

```
UpperLine = Average(High, UpperLength)  
LowerLine = Average(Low, LowerLength);
```

```
Plot1(UpperLine, "UpperLine");  
Plot2(LowerLine, "LowerLine");
```

Finally, declare any inputs:

```
Input: UpperLength(20), LowerLength(20);
```

```
Vars: UpperLine(0), LowerLine(0);
```

```
UpperLine = Average(High, UpperLength);  
LowerLine = Average(Low, LowerLength);
```

```
Plot1(UpperLine, "UpperLine");  
Plot2(LowerLine, "LowerLine");
```

Exercise: *15 Trailing Hi Lo

Learning objective: Using inputs; using the functions **Highest** and **Lowest**.

Description: This indicator plots two lines representing the highest value of an input and the lowest value of an input over the last n bars.

Create a new indicator and name it *#15 Trailing Hi Lo*. As stated, this indicator plots the highest and lowest values for a specified number of bars, creating a price channel on the chart. Therefore the indicator will require two plots. One plot will create a line which represents the highest High for the last n bars (in this case we'll use 8 and declare it as an Input). The other plot will create a line representing the lowest low for the last n bars.

First, declare and assign two variables, `HighLine` and `LowLine`, which will calculate the values to be plotted.

```
Vars: HighLine(0), LowLine(0);  
  
HighLine =  
LowLine =
```

Next, use the EasyLanguage Dictionary to find and drag in the already existing functions for `Highest` and `Lowest`. Each of these functions requires two parameters; *Price* and *Length*.

```
Vars: HighLine(0), LowLine(0);  
  
HighLine = Highest(  
LowLine = Lowest(
```

Declare the parameters for the functions in the assignment statements as inputs. For the *Price* parameter, use `HiPx` in the `Highest` function and `LoPx` in the `Lowest` function. Have these inputs default to `High` and `Low` respectively. Set the *Length* parameter in both assignment statements to `TrlgBars` and give this input a default value of 8.

```
Input: HiPx(High), LoPx(Low), TrlgBars(8);  
  
Vars: HighLine(0), LowLine(0);  
  
HighLine = Highest (HiPx, TrlgBars);  
LowLine = Lowest (LoPx, TrlgBars);
```

Finally, write the two plot statements and name them accordingly.

```
Input: HiPx(High), LoPx(Low), TrlgBars(8);

Vars: HighLine(0), LowLine(0);

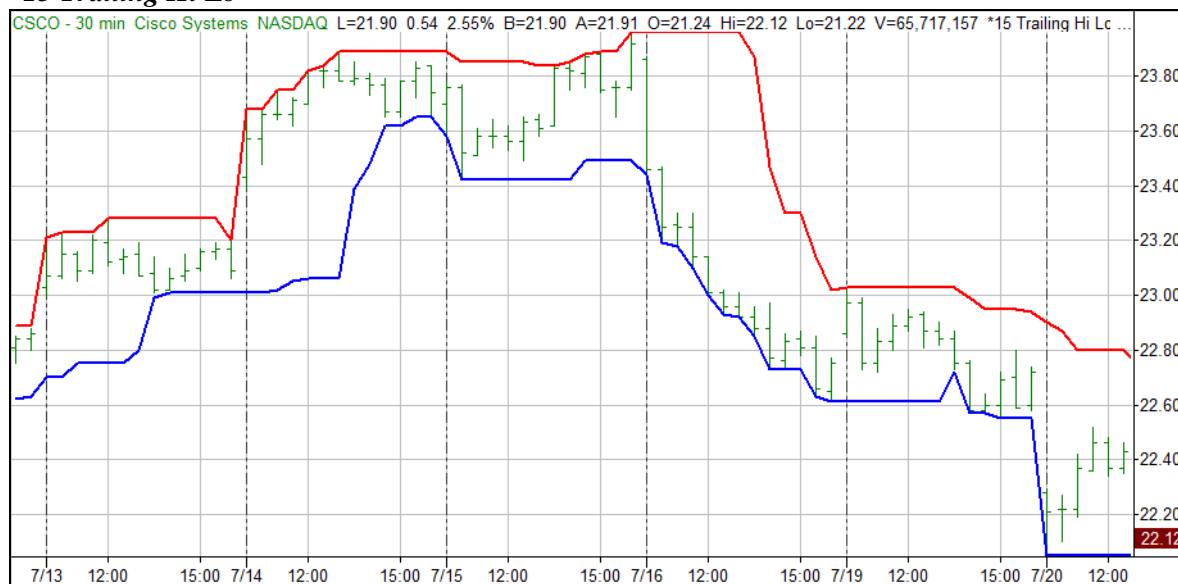
HighLine = Highest(HiPx, TrlgBars);
LowLine = Lowest(LoPx, TrlgBars);

Plot1(HighLine, "TrlgHigh");
Plot2(LowLine, "TrlgLow");
```

Verify this indicator and then make sure to set the properties for *Scaling to Same Axis as Underlying Data*. Don't forget to re-verify after exiting the *Properties* dialog.

Using the default input values, this indicator will now plot the highest high of the last 8 bars and the lowest low of the last 8 bars as two separate plots; recalculating and checking for new highs and new lows on each new bar. Inserting this indicator on a 30-minute chart should resemble something like the screenshot below:

***15 Trailing Hi Lo**



Question: Is there a point on the above chart where the bars penetrate the indicator? The answer is no. Let's examine why.

As an example, look at the red plot for *TrlgHigh*, which plots the *HighLine* variable. This variable is written to plot the highest high of the last 8 bars, and includes the current bar in the 8-bar count.

Question: Can the current bar ever have a higher high than itself? Obviously the answer is no. This is why the bars never penetrate the line. The same holds true for the plotted blue line, *TrlgLow*.

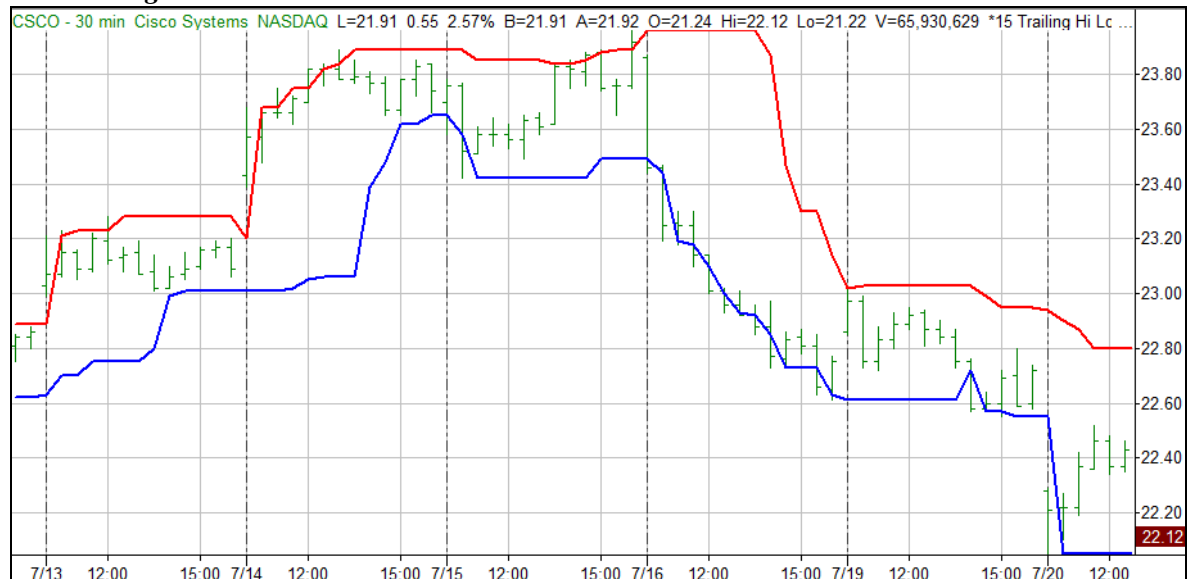
What you're looking for when using this indicator are **new** highs and lows. So, for most traders, the term, "The market made a new 8-bar high" would generally equate to a new high for the prior 8 bars, **not including the current bar**. The current calculation includes the current bar in the 8 bar comparison, but what is desired is to start the comparison 1 bar ago. To do this, modify the existing variable assignment statements to the following:

```
HighLine = Highest(HiPx, TrlgBars)[1];
LowLine = Lowest(LoPx, TrlgBars)[1];
```

Re-verify the indicator and now the chart should look like the figure below. Notice the bars do penetrate the lines.

Note: This concept would be more clearly illustrated, and more critical to get correct, if this were a strategy designed to buy and sell based on the bars penetrating the values of the plot lines. In the initial instance, trades would never be generated. Changing the EasyLanguage logic to begin the comparison starting one bar ago achieves the desired results.

****15 Trailing Hi Lo***



14. Relational Operators

True/false expressions require relational operators to compare two numeric values. The comparison returns either true or false. EasyLanguage has eight relational operators, which are available for use in analysis techniques and strategies. The six immediately below are probably obvious:

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
=	Equal to
<>	Not equal to

In addition to the above six, EasyLanguage has two unique relational operators that look for comparisons on consecutive bars:

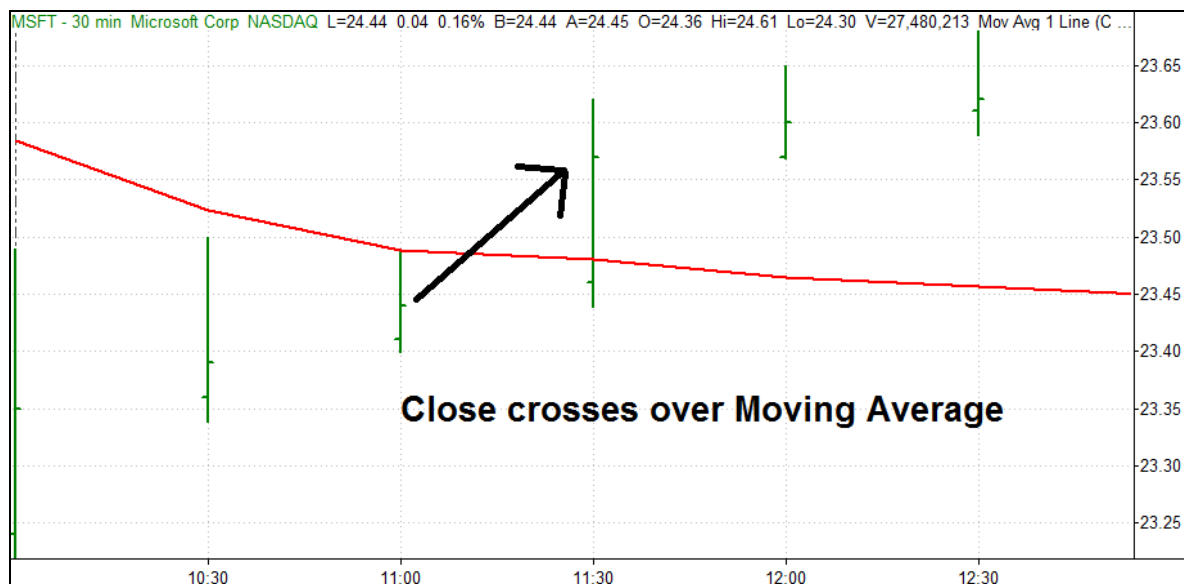
`Crosses over` (same as `Cross over` or `Crosses above` or `Cross above`)
returns True if

$x > y$ on current bar and $x < y$ previously

`Crosses under` (same as `Cross under` or `Crosses below` or `Cross below`)
returns True if

$x < y$ on current bar and $x > y$ previously

The two bars which occur at 11:00 and 11:30 on the chart below are an example of **Crosses over**, in this case for the close of the bar. As you can see, the bar with a time stamp of 11:00 has a close that is below the plotted moving average. However, the following bar, time-stamped 11:30, has a close which occurs above the moving average and thus has crossed over from one bar to the next.



True/false expressions use relational operators to compare values and make a true or false evaluation. Here are several examples:

```
Close = Open
```

The Close is equal to the Open (true), or it is not (false).

```
High > High [1]
```

The High is greater than the High of 1 bar ago (true), or it is not (false).

```
Date <> Date [1]
```

The Date is not equal to the Date of 1 bar ago (true), or it is (false).

```
Close crosses over Average (Close, 20)
```

The Close crossed over the 20-bar Average (true), or it did not (false).

15. If...Then Statements

An If...then statement allows the user to have an EasyLanguage instruction carried out only under certain conditions. Those conditions are contained in a true/false expression.

An If...then statement contains a true/false expression or expressions and an action to be taken.

```
If True/False Expression then  
    action to be taken;
```

For example:

```
If Close > Close[1] then  
    Plot1(High, "High");
```

Therefore, if the Close is greater than the Close of 1 bar ago then plot. Obviously, if this is not true, then no action (in this case Plot1) is taken.

16. Writing Alerts in EasyLanguage

Alerts are audio-visual or electronic notifications of specific market events as defined by the user.

- Indicators, and ShowMe and PaintBar studies can contain **alert** instructions; strategies cannot contain alert instructions. Strategies cannot use Alerts, but have different ways of notifying the user of buy and sell signals.
- Alerts are only triggered on the last bar of a chart, when criteria are true.
- The EasyLanguage instructions for the analysis technique must contain the alert criteria.
- Alerts must be enabled for the analysis technique, either by default setting in the EasyLanguage PowerEditor or in the *Format* dialog.

The last two bullet points above are extremely important when working with alerts, especially when writing your own into EasyLanguage documents. Think of it as two steps:

1. Placing a light bulb into a socket. This would represent writing the alert criteria into the EasyLanguage document.
2. Flipping the switch on the wall to turn on the light. This would represent enabling the alert.

Because an alert is usually contingent upon an event, EasyLanguage alert instructions are written using *If...then* statements.

```
If true/false expression then  
    Alert;
```

For example,

```
If Close > Highest(Close, 10)[1] then  
    Alert;
```

In English this translates into, “If the close is greater than the highest close of the last 10 bars starting 1 bar ago, then alert.”

In addition, in the second of the following two exercises, you’ll add custom alert messages to your EasyLanguage. Custom alert messages, which may include symbol name, are displayed in TradeStation’s Message Center when alerts are triggered.

Exercise: *16 Envelope2 Alert

Learning objective: Using `If...then` statements to set alert criteria.

Description: This is a re-write of **14 Envelope2* (which is a rewrite of Challenge 3, **13 Envelope*), incorporating conditions for alerts when the entire price bar is above the top line or below the bottom line.

Find and open your copy of *#14 Envelope2*. From the *File* menu in the Development Environment select *Save As...*, and rename the file *#16 Envelope2 Alert*. The EasyLanguage for this indicator is below, as well as how it should look on a chart:

```
Input: UpperLength(20), LowerLength(20);
```

```
Vars: UpperLine(0), LowerLine(0);
```

```
UpperLine = Average(High, UpperLength);
```

```
LowerLine = Average(Low, LowerLength);
```

```
Plot1(UpperLine, "UpperLine");
```

```
Plot2(LowerLine, "LowerLine");
```



Now, you will add alert criteria to this indicator to alert you when the entire bar is below the bottom line (LowerLine) or the entire bar is above the top line (UpperLine). How would you write this in EasyLanguage?

The two conditions, only one of which needs to be true to generate an alert, are as follows:

1. The entire bar is below the bottom line; which would mean the high of the bar would need to be less than the bottom line. In EasyLanguage this is:

$\text{High} < \text{LowerLine}$

2. The entire bar is above the top line; which would mean the low of the bar would need to be greater than the top line. In EasyLanguage this is:

$\text{Low} > \text{UpperLine}$

Add these as alert conditions to the existing EasyLanguage document:

```
Input: UpperLength(20), LowerLength(20);
```

```
Vars: UpperLine(0), LowerLine(0);
```

```
UpperLine = Average(High, UpperLength);
```

```
LowerLine = Average(Low, LowerLength);
```

```
Plot1(UpperLine, "UpperLine");
```

```
Plot2(LowerLine, "LowerLine");
```

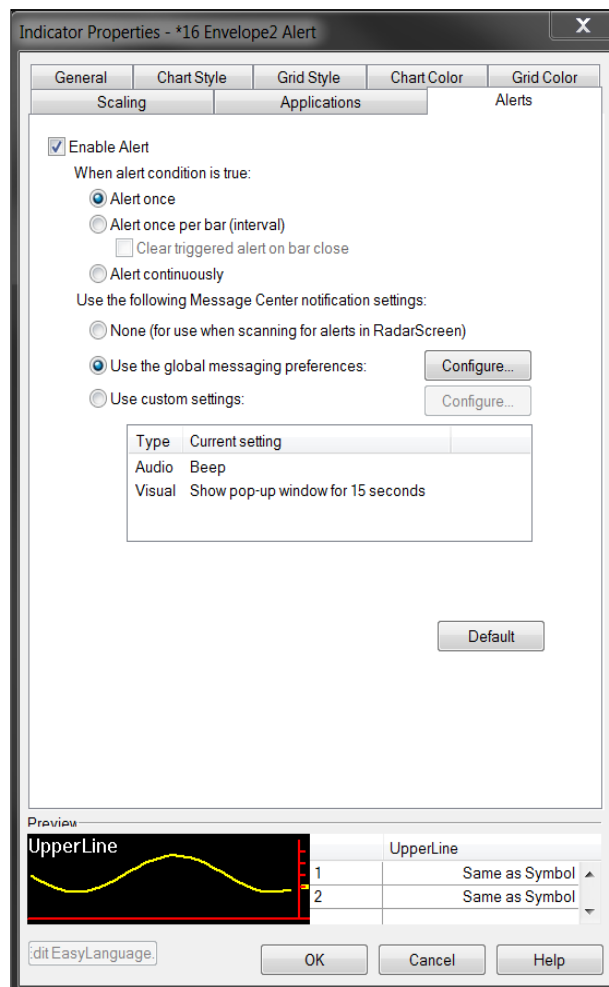
```
If Low > UpperLine then  
    Alert;
```

```
If High < LowerLine then  
    Alert;
```

Verify your work. Right click and select *Properties* to view the *Format Indicator* dialog. Select the *Alerts* tab. Click on the checkbox next to the option, *Enable Alert*. Also click on the radio button for *Alert Once* which will turn off this alert after the first time it is triggered and clear the *Enable Alert* checkbox.

There are many options and settings when using alerts in TradeStation. For detailed information, see the TradeStation Platform Help topic *Format Analysis Technique/Indicator - Alerts Tab* by clicking the *Help* button at the bottom right of the Format dialog while on the *Alerts* tab.

Click *OK* and verify again to save the changes. Insert the indicator on a chart. If either the low of the last bar is greater than the top line or the high of the last bar is less than the bottom line an alert will be triggered as shown in the screenshot below. If the alert does not trigger, you may try different symbols in an attempt to trigger the alert. In the following screenshot, you can see that the high of the last bar is below the lower line, thus triggering the alert.



***16 Envelope2 Alert**



Exercise: *17 Mov Avg & Bands

Learning objective: Using **If...then** statements to set alert criteria; review concepts for **Inputs, Variables and Functions**.

Description: This indicator plots 3 lines:

- A moving average
- The same moving average plus 1 standard deviation in price
- The same moving average minus 1 standard deviation in price

Alerts are set for price breaks through either of the outer bands, and, custom alert messages are included in the EasyLanguage.

Create a new indicator and name it **#17 Mov Avg & Bands**. Begin by writing the three plot statements as described above. The first plot, `Plot1`, is a simple moving average. The second plot, `Plot2`, is the same moving average plus one standard deviation in price. Finally, the third plot, `Plot3`, is the same moving average minus one standard deviation in price. The plot statements include not only the plot names, but the variables that will be declared and assigned below.

```
Plot1(MAvg, "Avg");
Plot2(HiBand, "HighBand");
Plot3(LoBand, "LowBand");
```

Now the variables used in the three plot statements need to be declared and assigned. *MAvg* should be assigned using the *Average* function. Also, declare a variable for the standard deviation, *Stdv*, which will be assigned shortly. *HiBand* should equal *MAvg* plus *Stdv* and *LoBand* should equal *MAvg* minus *Stdv*.

```
Vars: MAVg(0), Stdv(0), HiBand(0), LoBand(0);

MAvg = Average(Price, Length);
Stdv =
HiBand = MAVg + Stdv;
LoBand = MAVg - Stdv;

Plot1(MAvg, "Avg");
Plot2(HiBand, "HighBand");
Plot3(LoBand, "LowBand");
```

Next to the variable assignment statement, *Stdv*, you'll need to drop the function for standard deviation from the EasyLanguage Dictionary. Find the function, *StdDev*, and drag and drop this in to the Editor.

```
Vars: MAVg(0), Stdv(0), HiBand(0), LoBand(0);

MAvg = Average(Price, Length);
Stdv = StdDev(Price, Length);
HiBand = MAVg + Stdv;
LoBand = MAVg - Stdv;
```

```
Plot1(MAvg, "Avg");
Plot2(HiBand, "HighBand");
Plot3(LoBand, "LowBand");
```

Declare the `Price` and `Length` parameters used in both the `Average` and `StdDev` functions as inputs and give them default values of `Close` and `18`, respectively.

```
Inputs: Price(Close), Length(18);

Vars: MAvg(0), StdDev(0), HiBand(0), LoBand(0);

MAvg = Average(Price, Length);
StdDev = StdDev(Price, Length);
HiBand = MAvg + StdDev;
LoBand = MAvg - StdDev;

Plot1(MAvg, "Avg");
Plot2(HiBand, "HighBand");
Plot3(LoBand, "LowBand");
```

Now you're ready to add the alert criteria, which will be true in either of two instances. `TradeStation` will look for the closing price crossing over the `HiBand` or the closing price crossing under the `LoBand` and trigger alerts when either of these conditions is true. Adding the alert statements to the existing `EasyLanguage` should result in the following:

```
Inputs: Price(Close), Length(18);

Vars: MAvg(0), StdDev(0), HiBand(0), LoBand(0);

MAvg = Average(Price, Length);
StdDev = StdDev(Price, Length);
HiBand = MAvg + StdDev;
LoBand = MAvg - StdDev;

Plot1(MAvg, "Avg");
Plot2(HiBand, "HighBand");
Plot3(LoBand, "LowBand");

If Close crosses over HiBand
    then Alert ( "Close has crossed over HiBand " + Description );

If Close crosses under LoBand
    then Alert ( "Close has crossed under LoBand " + Description);
```

Notice the custom alert messages included in both alert statements, nestled within the parentheses following the word `Alert`. The text in quotes, as well as the symbol description, will appear in the Message Center and appropriate notifications.

Verify this indicator. Right click and select *Properties* to view the *Indicator Properties* dialog. Select the *Alerts* tab. As in the previous exercise, click on the checkbox next to the option, *Enable Alert*. Also click on the radio button for, *Alert Once*, which will turn off the alert after the first time it occurs and clear the *Enable Alert* checkbox.

Now click on the *Scaling* tab and set the scaling to *Same Axis to Underlying Data*, as the desired result is to have this indicator overlay the price bars. Re-verify the indicator after exiting the *Properties* dialog and place it on a chart.

*17 Mov Avg & Bands



17. ShowMe Studies

A ShowMe study marks each bar on a chart that meets the user's criteria. These studies are used for identifying historical occurrences, as well as monitoring for current occurrences, on a chart. Generally, ShowMe studies are most effective for identifying and marking sporadic events, i.e. those that occur on isolated bars. PaintBar studies, which you will get to shortly, are generally most effective for identifying conditions that persist on a series of bars.

Since ShowMe studies are generally not used to mark all the bars on a chart but rather only those on which some condition is true, or in other words, **if** something is true, this type of analysis technique will generally use `If...then` statements to identify those bars which should be marked.

Note: the format for the plot statement in a ShowMe study is identical to that used when writing an indicator.

```
PlotN(numeric expression, "plot name") ;
```

```
N = 1 to 99
```

An example of a simple plot statement would be:

```
Plot1(High, "MomentumUp") ;
```

The word `Plot1` in the sample plot statement above states that this is the first plot for this analysis technique, in this case a ShowMe study. The *numeric expression* to be plotted, in the sample statement `High`, is the location on the y-axis where the plot is to appear. In this case, the ShowMe will mark the high of the bar. `"MomentumUp"` is the plot name.

18. Block If...Then Statements

A block `If...then` statement is similar to a simple `If...then` statement but contains more than one action to be taken if the true/false expression is true. These actions are visually “blocked” together and are preceded and followed by the words `begin` and `end`, respectively. This tells TradeStation that if the conditional `If...then` is true, begin processing the actions to be taken and don’t stop until you see the word `end` (followed by a semicolon, of course).

```
If True/False Expression then begin
    action to be taken;
    action to be taken;
    action to be taken;
end;
```

In the following sample ShowMe study, you could attach each action to be taken to a separate true/false expression. For example:

```
If Open > High[1] then
    Plot1(Low, "GapUp");

If Open > High[1] then
    Alert;
```

Or using a block `If...then`, you could place a plot statement **and** an alert in a block `If...then`:

```
If Open > High[1] then begin
    Plot1(Low, "GapUp");
    Alert;
end;
```

This EasyLanguage statement tells TradeStation, “If the open of the current bar is greater than the high of 1 bar ago, then plot a ShowMe on the low of the current bar **and** alert”.

The block `If...then` adds efficiency to EasyLanguage. The first sample above using separate `If...then` statements forces the true/false expression, `Open > High[1]`, to be evaluated multiple times, once for each action to be taken. With a block `If...then` we attach multiple actions to one evaluation.

Exercise: *18 Wide Range

Learning objective: Writing a ShowMe study; using block **If...then** statements; using **Range** and **Highest** functions.

Description: This ShowMe marks those bars on which the **range** is greater than the **greatest range of the preceding 10 bars**.

Create a new ShowMe study named **#18 Wide Range**. (Remember to set the template in the *Select Template* dropdown to *None*.) In this exercise, since we now know how to use functions, we'll use the **Range** function to calculate the range (instead of **High - Low**, as we have been using up until now). We will also use the **Highest** function.

Type the following:

```
If Range >
```

Now use the EasyLanguage Dictionary to find the **Highest** function and drag it into the ShowMe. In the *Search* box of the Dictionary type "Highest" and click the arrow to the right. Select the function *Highest* from the list. Drag and drop this function into the ShowMe.

```
If Range > Highest (
```

The **Highest** function calls for two parameters, one for **Price** and one for **Length**. Since you are looking for the **Highest Range** of the last 10 bars, you can use the function **Range** for the **Price** parameter and 10 for the **Length** parameter. However, it might be a good idea to use an input for the **Length**.

```
Input: TrlgBars (10);  
If Range > Highest(Range, TrlgBars)
```

You also need to add a reference for data from a previous bar since the current **Range** can never be higher than itself:

```
Input: TrlgBars (10);  
If Range > Highest(Range, TrlgBars)[1]
```

What we now have is, "If the current range is greater than the highest range of the last 10 bars starting 1 bar ago..."

Now add the actions to be taken inside of a block **If...then**. Remember to use **begin** and **end** (followed by a semicolon) around the actions to be taken.

```
Input: TrlgBars(10);  
If Range > Highest(Range, TrlgBars)[1] then begin  
    Plot1(Close, "Wide Range");  
    Alert;  
end;
```

Verify the ShowMe and place this on a chart. The ShowMe study marks the bars on the chart where the range is greater than the greatest range of the last 10 bars.

***18 Wide Range**



19. If...Then...Else and NoPlot Statements

An If...then...else statement includes an action to be taken if the true/false condition is true and a different action to be taken if the condition is false.

```
If True/False Expression then
    action to be taken
else
    alternative action to be taken ;
```

An If...then...else statement along with a NoPlot statement is often used in a ShowMe study to remove the mark from a bar if a condition was true during the bar and becomes false later in the same bar or at bar close.

```
If Close > Close[1] then
    Plot1( Low, "UpBar")
else
    NoPlot(1) ;
```

In this case, as an example, if the Close is greater than the Close of 1 bar ago, the ShowMe plot will appear on the chart. However, including the NoPlot statement will cause the Plot to be removed if the condition becomes false at some point later in the bar or at bar close. The number 1 in parentheses which follows the NoPlot statement, instructs TradeStation to remove Plot1.

Notes: Although using a NoPlot statement will remove a plot that was previously true, any active alerts are monitored in real time. NoPlot will not remove a plot from a previously closed bar, only from the current bar as the conditions change.

Block If...then statements may also be written as block If...then...else statements.

```
If True/False Expression then begin
    action to be taken;
    action to be taken;
    action to be taken;
end
else begin
    action to be taken;
    action to be taken;
    action to be taken;
end;
```

Note: the first "end" statement does not require a semicolon after it because it is not the end of the entire "if" statement.

Challenge 4: *19 Weak Close

Learning objective: Writing a ShowMe study; using block `If...then...else` and `NoPlot` statements.

Write a ShowMe that marks the high of those bars that have a close that is in the bottom third of their range. Include a NoPlot statement to remove a Plot if a true condition becomes false before bar close.

Include instructions to be alerted when this condition occurs.

The range of the bar is `High - Low`, but there is also a function called `Range` that makes this calculation for you.

20. PaintBar Studies

A PaintBar study “paints” all or part of each bar on a chart that meets the study’s criteria. These studies are used for identifying historical occurrences, as well as for monitoring current occurrences, on a chart. Generally, PaintBar studies are most effective for identifying conditions that persist on a series of bars. Their usage differs from ShowMe studies, with which you worked earlier, and which are best used to identify sporadic events that occur on isolated bars.

A special style of plot statement may be used for PaintBar studies.

```
PlotPB(StartPaint, EndPaint, "plot name");
```

StartPaint and *EndPaint* refer to the start and end points of the painting process on the y-axis. Normally PaintBar studies paint the entire bar, from low to high, but are certainly not limited to this style. You have the option of painting portions of the bar, for example, from the open to the close.

As with other analysis techniques, including “*plot name*” is optional, but recommended. Other optional parameters for painting and formatting are also available. Also, as we examined and used with an earlier ShowMe study, the reserved word `NoPlot(n)` may be used in a PaintBar study to remove a plot if a condition for plotting were true at some earlier point during the bar and then becomes false.

Exercise: *20 MomentumPositive

Learning objective: Writing a PaintBar study using the `PlotPB` statement.

Description: This PaintBar marks those bars on which the momentum is greater than 0 and alerts if this condition is true. A `NoPlot` statement is used as well.

In the Development Environment, use the *File – New* menu sequence to create a new PaintBar study named **#20 MomentumPositive**. Remember to set the template in the *Select Template* dropdown to *None*. In the EasyLanguage Editor, create a block *If...then* for the condition “momentum greater than 0,” that includes instructions to plot a PaintBar and alert. Since this is a block *If...then*, you will need to remember to use `begin` and `end` to bracket the `Plot` and `Alert` instructions. We also add the `NoPlot(1)` statement following `else`:

```
If Mom > 0 then begin
    PlotPB(High, Low, "MomPos");
    Alert;
end
else
    NoPlot(1);
```

Create variable declaration and assignment statements for *Mom*:

```
Vars: Mom(0);

Mom = Momentum( Price, Length);

If Mom > 0 then begin
    PlotPB(High, Low, "MomPos");
    Alert;
end
else
    NoPlot(1);
```

Now specify inputs for *Price* and *Length* and you're done:

```
Input: Price(Close), Length(10);

Vars: Mom(0);

Mom = Momentum( Price, Length);

If Mom > 0 then begin
    PlotPB(High, Low, "MomPos");
    Alert;
end
else
    NoPlot(1);
```

Verify your work and plot this PaintBar to identify those bars on which momentum is greater than 0.

***20 Momentum Positive**



21. Referencing Bar Date in EasyLanguage

Every bar on a chart, regardless of interval or type, is marked with its ending date. EasyLanguage has its own numeric format for dates. Below is an example which represents the date December 9, 2003. Weekly charts always end on Friday and monthly charts always end on the last trading day of the month.

1031209

103 represents the year 2003 with the number 100 representing 2000. 0 would represent 1900. For example 000 equates to the year 1900 and 088 would represent the year 1988.

12 represents the month, which is December.

09 represents the date, in this case, the 9th of the month

The reserved word `Date` returns the date of the bar in this format. There are many functions and reserved words to help you work with this information.

These dates may be referenced to find or use specified dates or days of the week and are also used as markers, for example, to denote one day from the next on an intraday chart.

```
Date <> Date[1]
```

This true/false expression is true on the first bar of each day on an intraday chart. (Note: this example is most useful only on a chart with a session that begins and ends within the same calendar day.)

Exercise: *21 MyDay

Learning objective: Writing a PaintBar study; referencing Date in EasyLanguage.

Description: This ShowMe marks each bar that occurs on the day of the week specified by the input.

Create a new PaintBar and name it **#21 MyDay**. As stated in the description, this PaintBar study will mark those bars on the chart which occur on a specified day of the week. You will use an input to specify the day of the week and an `If...then` statement to determine a true condition which will then cause the study to paint the bar.

In the Editor, type the following and don't forget to verify when you are done:

```
Input: DoW (Wednesday) ;

If DayOfWeek(Date) = DoW then
    PlotPB(High, Low, "DoW");
```

The `If...then` statement uses the `DayOfWeek` reserved word to return the day of the week for each bar on the chart. More specifically, `DayOfWeek` returns a value for each bar on the chart based on the date for that bar. The returned value corresponds to a specific day of the week as follows:

- 0 = Sunday
- 1 = Monday
- 2 = Tuesday
- 3 = Wednesday
- 4 = Thursday
- 5 = Friday
- 6 = Saturday

For those bars that return the value 3, which is equal to Wednesday, TradeStation marks the bars by painting them. Insert **#21 MyDay** on your chart. This PaintBar may be used on daily and intraday charts. You may want to change the chart interval to see how the PaintBar looks on each type of chart.

***21 MyDay**



22. Referencing Bar Time in EasyLanguage

Every bar in a chart is stamped with its ending time, again regardless of interval or type. The time is referenced in EasyLanguage according to military time, with no colon. The reserved word `Time` returns the time of the bar in this format:

```
Time > 1500
```

This true/false expression is true on every bar that ends after 3:00 pm.

There are many functions and reserved words to help you work with this information. See the Dictionary for more information.

23. Logical Operators

Logical operators are used to connect multiple true/false expressions into longer true/false expressions. The choice of logical operator determines whether only some or all of the individual true/false expressions must be true for the entire expression to be true.

AND both expressions must be true for the entire expression to be true

`High > High[1] AND Close > Close[1]`

OR if either expression is true then the entire expression is true

`High > High[1] OR Close > Close[1]`

Logical operators are evaluated from left to right, with AND having precedence over OR. Parentheses should be used to define the order of evaluation.

Challenge 5: *22 Parts Of Day

Learning objective: Writing a PaintBar study using Time and logical operators

Write a PaintBar that paints the lower half of those bars that are in the first and last hours of the day.

Declare Inputs for the times that delineate the first and last hours of the day.

24. Strategies

At this point, you have written indicators, and ShowMe and PaintBar studies. You should now be ready to apply what you have learned and begin to create your own trading strategies. A trading strategy monitors the market for past and current occurrences of criteria that are position entry and exit points. These occurrences are indicated on a chart and logged for performance reporting purposes. Current occurrences of trading criteria may also be sent to the marketplace for actual execution using TradeStation's strategy automation features.

The process of strategy development begins with technical analysis. The first steps are visual and, largely, what we have focused on up until this point in this book. To develop strategies, one first needs to be able to use analysis techniques (either those written by others or yourself) to identify trading opportunities. A comfort level needs to be established in regard to being able to look at a chart and determine what's going on with market direction and indicators. Are there identifiable correlations that are evident? If so, then the next logical step for the strategy trader is to ask the question, "What if I write and test a set of buy and sell rules based on these correlations?"

Each strategy written in EasyLanguage generally contains at least one trading action. Multiple strategies, each with any number of trading actions, may be applied to a Chart Analysis Window simultaneously.

EasyLanguage Order Syntax for Strategies

EasyLanguage uses four trading verbs to identify the market action to be taken in a strategy:

- **Buy:** establish, or add to, a long position (any existing short position will be covered before a long position is established).
- **SellShort:** establish, or add to, a short position (any existing long position will be liquidated before a short position is established).
- **Sell:** sell to liquidate a long position only.
- **BuyToCover:** buy to cover a short position only.

Note: These four verbs act as described above regardless of the symbol to which the strategy is applied. That is, the verb `SellShort` must be used to generate short positions even if the strategy is applied to a futures or forex symbol.

These actions may be taken in the following ways:

- `Next bar at market` = on the Open of the next bar.
- `Stop` = on the next bar if the stop is triggered.
- `Limit` = on the next bar if the limit price is reached.
- `This bar on close` = on the Close of this bar (for backtesting only).

Proper order syntax combines the trading verbs followed with the order type as such:

```
AnyVerb next bar at market;  
AnyVerb next bar at AnyPrice limit;  
AnyVerb next bar at AnyPrice stop;  
AnyVerb this bar on close;
```

Here are several sample statements, which include order statements:

```
If true/false expression then  
    Sell Short next bar at market;  
  
Sell next bar at Close of this bar + .10 limit;  
  
Buy to Cover next bar at High of this bar + .05 stop;
```

Limit and Stop orders may also reference the Open of the next bar in the limit or stop prices.

```
Buy next bar at Open of next bar - .05 limit;
```

Exercise: *23 Breakout

Learning objective: Writing a strategy; stop order syntax.

Description: This strategy buys (long) on a breakout above the highest high of the preceding 8 bars; it sells short on a break down below the lowest low of the preceding 8 bars.

Create a new strategy named *#23 Breakout*. Make sure to select *None* from the *Select Template* dropdown. The instructions indicate that this strategy buys and sells short based on highest and lowest prices of the last 8 bars. First, declare and assign variables for the buy and sell prices:

```
Vars: BuyPx(0), SellPx(0);  
  
BuyPx = Highest(High, 8) + .02;  
SellPx = Lowest(Low, 8) - .02;
```

Now, examine what you have typed. The variables `BuyPx` and `SellPx` are declared and assigned to represent the highest high of the last 8 bars and the lowest low of the last 8 bars, respectively. The addition or subtraction of .02 is to be sure that price penetrates the 8-bar high or low before taking the signal. The `Highest` and `Lowest` functions, which may be found in the EasyLanguage Dictionary, are used. Next, add the `Buy` and `Sell` statements:

```
Vars: BuyPx(0), SellPx(0);  
  
BuyPx = Highest(High, 8) + .02;  
SellPx = Lowest(Low, 8) - .02;  
  
Buy ("Brk LE") next bar at BuyPx Stop;  
Sell Short ("Brk SE") next bar at SellPx Stop;
```

Notice the entry names `Brk LE` and `Brk SE`, placed inside a set of parentheses and quotations. These names will appear on a chart when the strategy is applied. Naming entries and exits is useful when a strategy contains multiple entries and/or exits based on several different conditions. This practice allows you to look at buys and sells on a chart and determine which entry or exit was triggered for a particular bar.

One final item: it may be useful to set the length parameter in both variable assignment statements as an input. Therefore,

```
BuyPx = Highest(High, 8) + .02;  
SellPx = Lowest(Low, 8) - .02;
```

becomes

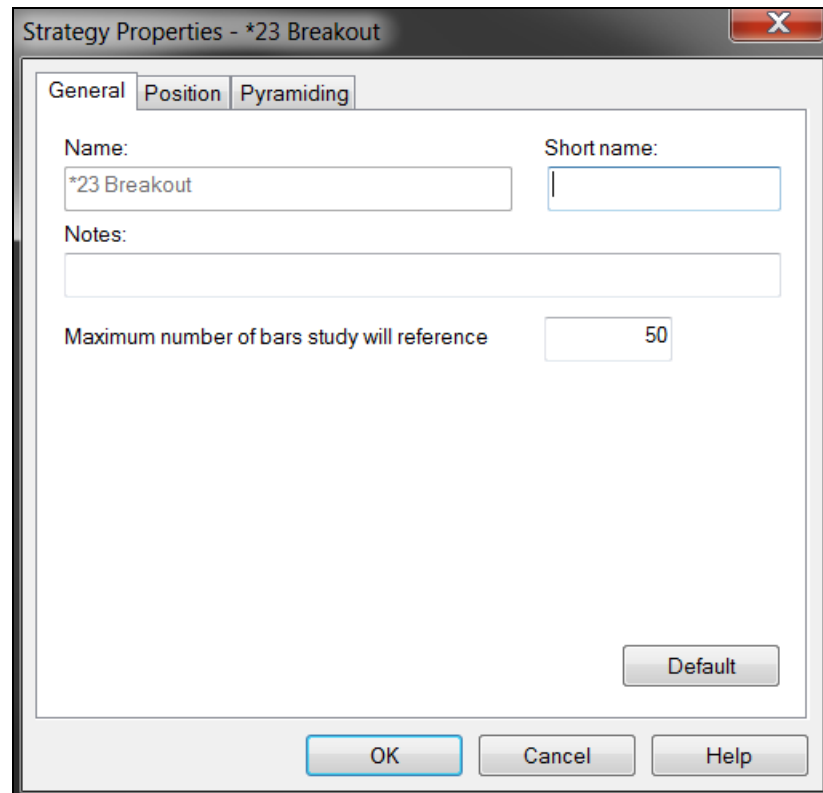
```
Input: Length(8);  
Vars: BuyPx(0), SellPx(0);  
  
BuyPx = Highest(High, Length) + .02;  
SellPx = Lowest(Low, Length) - .02;  
  
Buy ("BrkBuy") next bar at BuyPx Stop;  
Sell Short ("BrkSS") next bar at SellPx Stop;
```

It is important to note that in this exercise, the order statements are not placed after a conditional. This is because stop orders are inherently conditional. They're conditional based on the stop price being achieved.

Also, if you look at the variable assignment statements you'll notice that we do not refer to "1 bar ago", or [1], as we did earlier in other created studies. This is because the stop orders are entered for the **next** bar. So, we want to use the highest high of the last 8 bars, including the current bar, to enter an order for the next bar.

Verify your work and access the *Strategy Properties* dialog by right-clicking in the Editor and selecting *Properties* from the shortcut menu. This dialog differs slightly from those for indicators, ShowMe's and PaintBars. Typically, there are no properties to set in a strategy.

For example, you can see from the figure below that there are only three tabs in this dialog, *General*, *Position*, and *Pyramiding*.



The *General* tab contains a field called *Maximum number of bars the study will reference*, with a default setting of 50. This determines the maximum look back period for the strategy as it moves across the chart. In other words, as the strategy is moving bar by bar across the chart, the furthest back it can reference is 50 bars (by default). You might need to increase this setting if your strategy references data from more than 50 bars ago.

Note: Indicator, ShowMe and PaintBar studies also have a setting for *Maximum number of bars study will reference*. However, this defaults to *Auto-detect* and rarely needs to be changed for analysis techniques.

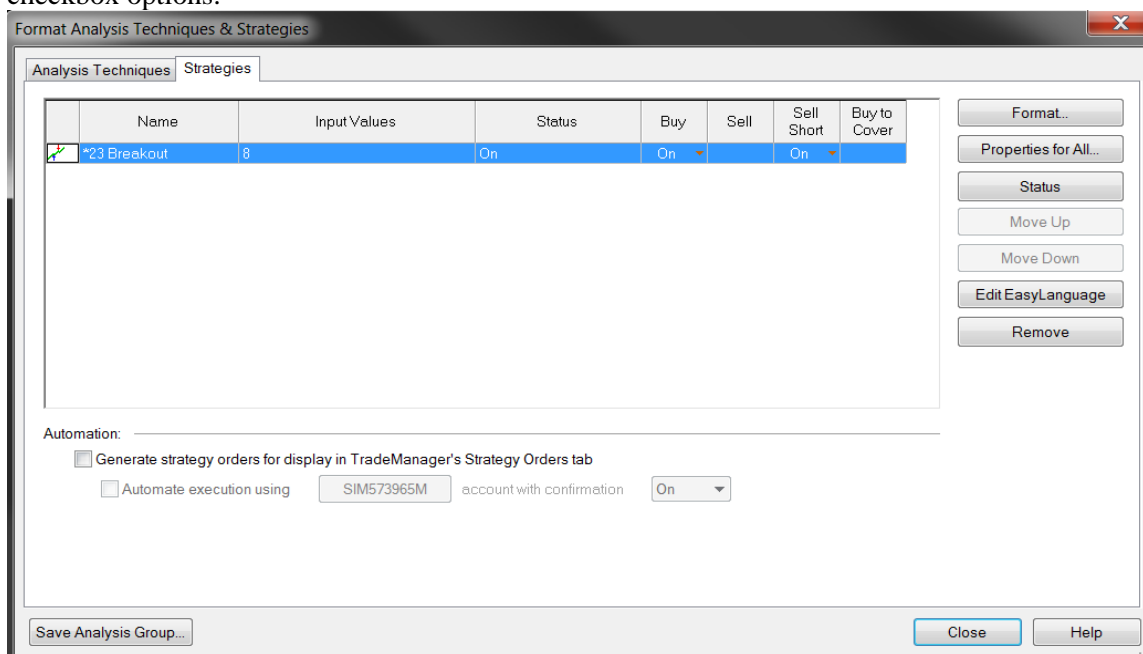
The *Position* tab allows you to modify settings for maximum open entries and contract/shares per position, while the *Pyramiding* tab provides options for modifying pyramiding settings. For detailed information on these tabs please refer to the TradeStation Development Environment Help, accessible from the Help menu within the TradeStation Development Environment, or by clicking on the *Help* button in this dialog.

Remember to re-verify the strategy after exiting the dialog.

Return to a Chart Analysis window in the TradeStation Platform. Insert the indicator **#15 Trailing Hi Lo**. This indicator displayed a price channel which is the conceptual basis for this strategy. You will find it useful to see both the indicator lines and the strategy trades displayed on the same chart. This is an important aspect of visualizing and developing strategy ideas.

Now apply the strategy to the chart. Click *Insert* then *Strategy...* from the main menu. In the *Insert Strategies* dialog, make sure the *Prompt for Format* checkbox is selected then click *OK*. You will immediately see the *Format Analysis Techniques and Strategies* dialog.

First, let's look at the important automation options at the bottom of the dialog. There are 2 main checkbox options:



The first checkbox is *Generate strategy orders for display in TradeManager's Strategy Orders tab*. Use this to have the strategy log any orders it generates in the *Strategy Orders* tab in the TradeManager **without** actually sending any orders to the marketplace.

The second checkbox, *Automate execution using [account number]...* automatically sends any orders generated by this strategy to the marketplace for the selected account. This option should only be used when you have tested and refined a strategy and are fully ready to automate order execution. You should automate your strategy for some period of time in the simulator before automating with real dollars.

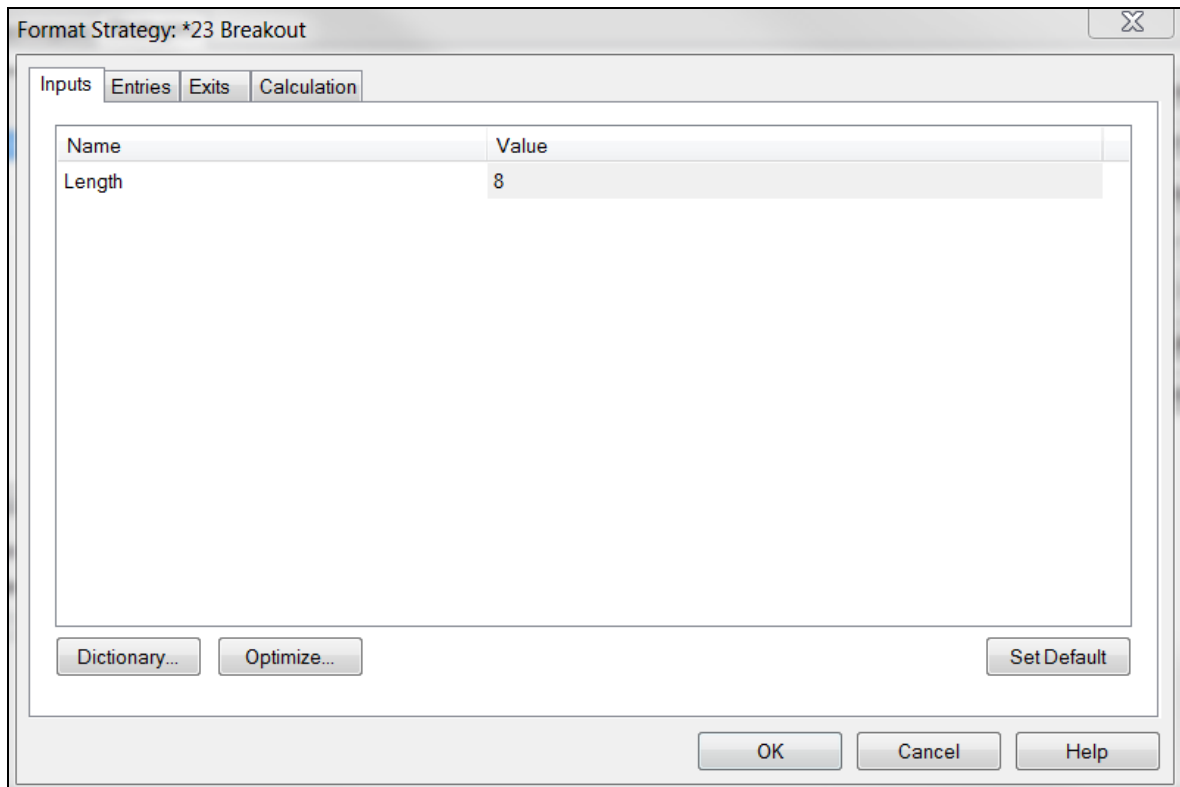
For detailed information, please access the corresponding documentation in the TradeStation Help by clicking on the *Help* button at the bottom right of this dialog.

Note: These automation settings may be used with live and simulated accounts.

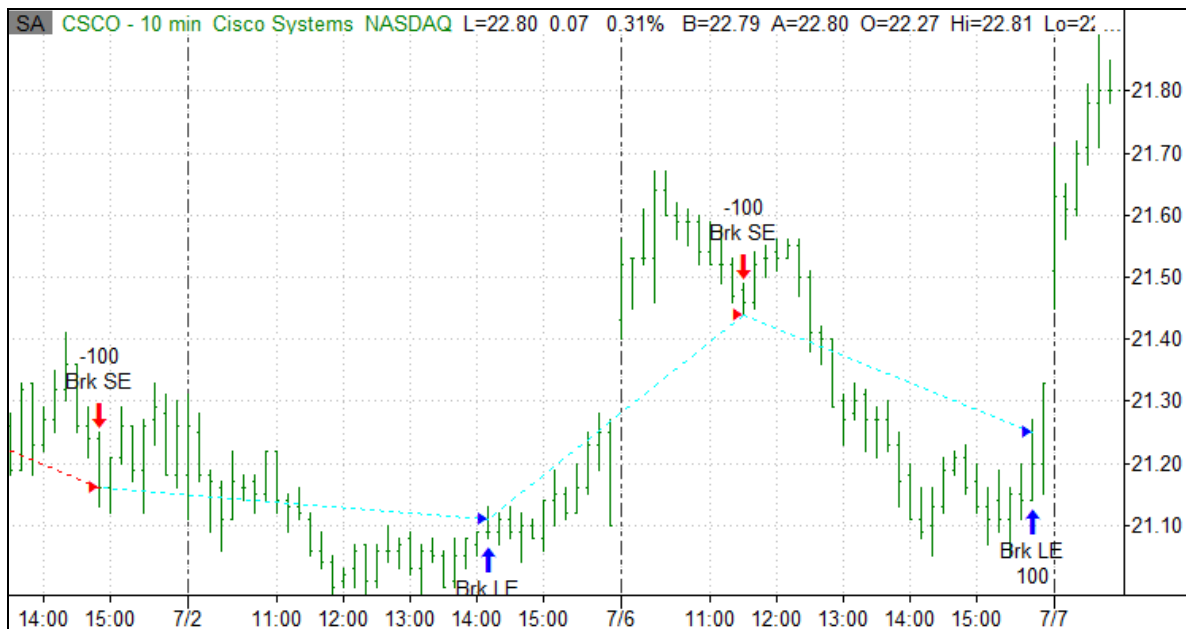
For the purposes of this course, you should not select any automation settings.

Clicking on the *Format* button opens the *Format Strategy* dialog. This dialog is different than the *Strategy Properties* dialog you accessed earlier (from the EasyLanguage Editor) when creating the strategy. It contains four tabs; one for modifying and optimizing inputs, two for customizing how entries and exits appear visually on a chart, and one tab for enabling Intraday Order Generation. For detailed information on any of these tabs, please access the corresponding

documentation in the TradeStation Help by clicking on the *Help* button at the bottom right of the dialog.



***23 Breakout**



Strategy Properties

The *Properties for All...* button in the *Format Analysis Techniques and Strategies* dialog takes you to the *Strategy Properties for All Strategies on this Chart* dialog. This dialog contains three tabs: the *General* tab, the *Backtesting* tab and the *Automation* tab. Many details for all of the strategies appearing in this particular Chart Analysis window can be customized here, such as slippage and commissions, or specifications on trade size and strategy order fill logic. Again, for detailed information on any of these tabs please access the corresponding documentation in the TradeStation Help by clicking on the *Help* button at the bottom right of the dialog.

Strategy Properties for All Strategies on this Chart

General Backtesting Automation

Currency
Base currency of: Symbol (US Dollar)

Costs/Capitalization
Commission: \$ 0 ☒ per Trade ☐ per Share/Contract
Slippage: \$ 0 ☒ per Trade ☐ per Share/Contract
Initial Capital: \$ 100,000
Interest Rate: 2 %
Note: Initial Capital and Interest Rate are used only in the Strategy Performance Report.

Back-testing resolution
☐ Use Look-Inside-Bar Back-testing
☐ Tick 1 ticks
☒ Intra-day 1 minute
☐ Daily
Maximum number of bars study will reference 50

Position limits (for pyramiding strategies only)
☐ Allow up to 50 entry orders in the same direction as the currently held position:
☐ when the order is generated by a different entry order
☒ regardless of the entry that generated the order
Maximum shares/contracts/units per position 65,000

Trade size (if not specified by strategy)
☒ Fixed Shares/Contracts/Units 100
☐ Dollar(s) per trade \$ 10,000
Round down to nearest 100 shares/contracts/units
Minimum number shares/contracts/units: 100

Note: For exercise 23 trade size is set to 100 shares.

Return to the *Format Analysis Techniques and Strategies* dialog and click Close to apply this strategy to the chart. Upward pointing blue arrows represent long entries while downward pointing red arrows represent short entries. The number that appears below each of these words is the net position at the close of the bar. Notice the entry names *Brk LE* and *Brk SE* labeled on the chart. *Buy* and *Short* are used by default for entries but were customized using *Brk LE* and *Brk SE*. Likewise, *Cover* and *Sell* are used for exits and may also be customized. Incidentally, there are no exits in this strategy since it continually closes and then reverses the open position.

To view performance information on the strategy, click *View* then *Strategy Performance Report* from the main menu.

Exercise: *24 Mov Avg Cross

Learning objective: Writing a strategy; using strategy position reserved words.

Description: This strategy takes long and short positions based on the crossovers of two moving averages. A position is closed when it has been held at least a minimum number of bars and is not making at least a minimum profit.

Create a new strategy named *#24 Mov Avg Cross*. First, write the variable declaration and assignment statements for the 2 moving averages.

```
Vars: ShortMA(0), LongMA(0);  
  
ShortMA = Average(Close, ShortLen);  
LongMA = Average(Close, LongLen);
```

The *Length* parameters ShortLen and LongLen are intended to be inputs. Write the input declaration statement and set the default values to 9 and 18, respectively.

```
Input: ShortLen(9), LongLen(18);  
  
Vars: ShortMA(0), LongMA(0);  
  
ShortMA = Average(Close, ShortLen);  
LongMA = Average(Close, LongLen);
```

Now that the calculations have been established, it's time for the entry commands:

```
Input: ShortLen(9), LongLen(18);  
  
Vars: ShortMA(0), LongMA(0);  
  
ShortMA = Average(Close, ShortLen);  
LongMA = Average(Close, LongLen);  
  
If ShortMA crosses over LongMA then  
    Buy next bar at market;  
If ShortMA crosses under LongMA then  
    SellShort next bar at market;
```

Finally, we add the exit rules. As described above, we will exit if it seems that the signal is not following through. We will gauge this by how long we have been in the position, measured in bars, and the profitability of the position. Reserved words *BarsSinceEntry* and *OpenPositionProfit*, which may be found in the Dictionary, are used to track the position status.

```
Input: ShortLen(9), LongLen(18);

Vars: ShortMA(0), LongMA(0);

ShortMA = Average(Close, ShortLen);
LongMA = Average(Close, LongLen);

If ShortMA crosses over LongMA then
    Buy next bar at market;
If ShortMA crosses under LongMA then
    SellShort next bar at market;

If BarsSinceEntry > MinHold and OpenPositionProfit < MinProf
then begin
    Sell next bar at market;
    BuyToCover next bar at market;
end;
```

Note the words MinHold and MinProf; these should be declared as inputs.

```
Input: ShortLen(9), LongLen(18), MinHold(8), MinProf(50);

Vars: ShortMA(0), LongMA(0);

ShortMA = Average(Close, ShortLen);
LongMA = Average(Close, LongLen);

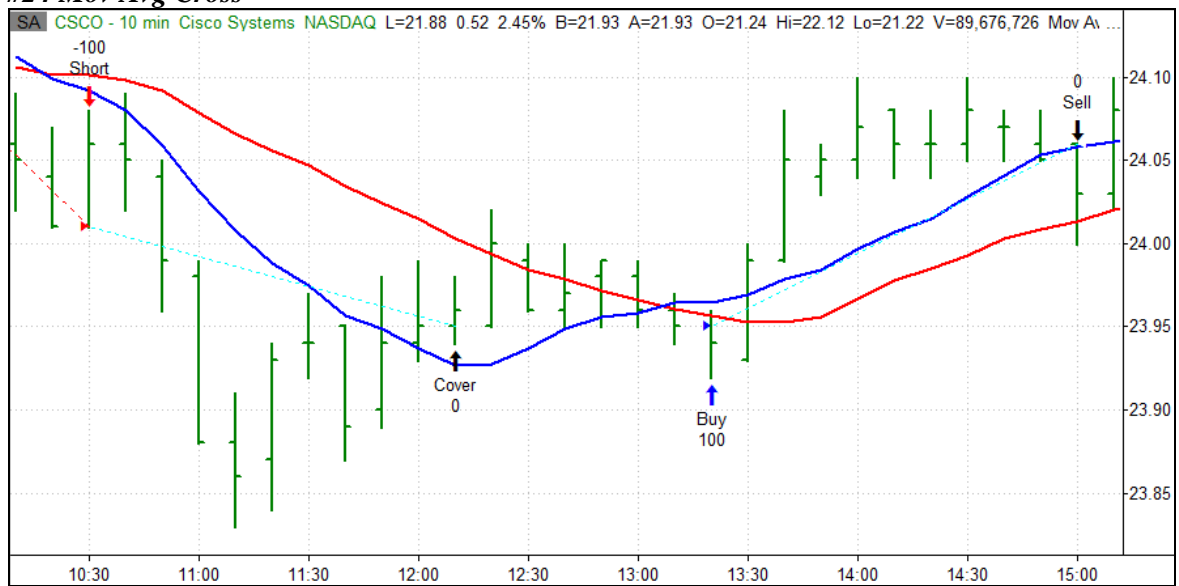
If ShortMA crosses over LongMA then
    Buy next bar at market;
If ShortMA crosses under LongMA then
    SellShort next bar at market;

If BarsSinceEntry > MinHold and OpenPositionProfit < MinProf
then begin
    Sell next bar at market;
    BuyToCover next bar at market;
end;
```

Verify the strategy.

Return to a Chart Analysis window in the TradeStation Platform. Insert the indicator Mov Avg 2 Lines, with the default input values of 9 and 18 for the lengths. As in the previous exercise and with strategies generally, you will find it useful to see both the indicator and the strategy trades displayed on the same chart to help you see the strategy concept. Now apply the strategy to the chart.

#24 Mov Avg Cross



25. True/False Variables

A true/false variable is a true/false expression given a name for easy reference. The name may then be used to reference that true/false expression repeatedly within a single analysis technique or strategy without re-typing the expression. This improves the efficiency of EasyLanguage because the expression needs to be evaluated only once for each bar and may then be referenced by variable name.

True/false variables are used to

- Improve processing efficiency
- Reduce typographical errors
- Increase readability

Note: Each variable is only valid within the analysis technique or strategy in which it is declared and/or assigned.

User-declared True/False Variables

Exactly parallel to user-declared numeric variables, EasyLanguage permits **user-declared true/false variables**. This provides for the creation of custom names for true/false variables. These names should be descriptive of the expression assigned to that variable.

Just as with user-declared numeric variables, user-declared true/false variables first need to be declared. However, unlike user-declared numeric variables which are initialized to a numeric value, user-declared true/false variables are initialized to either TRUE or FALSE. Generally, they are initialized to false. Why? The reason is that traders generally think in terms of conditions that must be true in order to place trades, not conditions that must be false in order to place trades. Remember, you'll also need to assign a value (using a variable assignment statement) to any user-declared true/false variable before it is used.

Variable Declaration Statement

This statement declares a true/false variable named UpBar with an initial value of FALSE:

```
Variables: UpBar(false);
```

Multiple true/false variables may be declared in the same declaration statement. In fact, numeric and true/false variables may be declared in one declaration statement. The following statement declares two true/false and one numeric variable:

```
Variables: NetChange(0), UpBar(false), DownBar(false);
```

Variable Assignment Statement

This statement assigns the true/false expression for UpBar to the variable:

```
UpBar = Close > Close[1];
```

Remember, just as with user-declared numeric variables, user-declared true/false variables require a two-step process: first declaration, then assignment. Also, remember that although multiple variables may be declared in one declaration statement, each variable must have its own assignment statement.

Exercise: *25 Momentum Cross

Learning objective: Using user-declared true/false variables; **MRO** function.

Description: This strategy buys (long) when momentum crosses over 0, as long as it has not crossed under 0 within the last 4 bars; it sells short when momentum crosses under 0, as long as it has not crossed over 0 within the last 4 bars; it sells (liquidates long positions) if momentum declines on 2 consecutive bars; it buys to cover (covers short positions) if momentum rises on 2 consecutive bars.

Create a new strategy named *#25 Momentum Cross*. First write the exits as explained in the instructions above:

```
If Mom < Mom[1] and Mom[1] < Mom[2] then
    Sell next bar at market;

If Mom > Mom[1] and Mom[1] > Mom[2] then
    BuyToCover next bar at market;
```

The first statement sells (to liquidate long positions) if the momentum of the current bar is less than the momentum of 1 bar ago **and** the momentum of 1 bar ago is less than the momentum of two bars ago. In other words, a sell is generated if momentum declines on two consecutive bars.

The second statement buys to cover if the momentum of the current bar is greater than the momentum of 1 bar ago **and** the momentum of 1 bar ago is greater than the momentum of two bars ago. In other words, a buy to cover is generated if momentum rises on two consecutive bars.

Now you'll write the entry statements. According to the exercise instructions there are two possibilities: one that will cause the strategy to buy and one to sell short. In order to buy, momentum needs to cross over 0 (let's call this a **BullCx**) **and** must not have crossed under 0 within the last four bars. In order to sell short, momentum needs to cross under 0 (let's call this a **BearCx**) **and** must not have crossed over 0 within the last four bars.

To write the two entry statements, you will use the **MRO** function, which you can find in the EasyLanguage dictionary. **MRO** stands for **M**ost **R**ecent **O**ccurrence. The **MRO** function is specifically designed to identify **when** a certain condition last occurred. This function requires three parameters:

```
MRO (Test, Length, Instance)
```

Test is the true/false expression to test for. **Length** is the number of trailing bars to check and includes the current bar. **Instance** indicates which instance to check for; that is, 1 = most recent occurrence, 2 = 2nd most recent occurrence, etc.

The function returns either a value that is equal to the number of bars ago the true/false expression was true or returns a value of -1 if the true/false expression was not found to be true within the last **Length** bars.

For example:

```
MRO(Close > Open, 3, 1)
```

This statement checks for the most recent occurrence of the `Close` being greater than the `Open` for the last three bars (including the current one). If the `Close` was not greater than the `Open` over the last three bars, a value of `-1` would be returned, meaning this didn't occur. If the `Close` was greater than the `Open` on the bar before the current bar, then a value of `1` would be returned. If the `Close` was greater than the `Open` on the current bar then a value of `0` would be returned, indicating the current bar is the most recent occurrence of the true/false expression.

The two entry statements, along with variables declared and assigned for `BullCx` and `BearCx` as well as `Mom`, can be written as follows and added to the already written exits. Also, it will be useful to declare an input for the `Length` parameter in the `Momentum` function assigned to the variable `Mom`.

```
Input: Length(10);

Vars: Mom(0), BullCx(false), BearCx(false);

Mom = Momentum(Close, Length);
BullCx = Mom crosses over 0;
BearCx = Mom crosses under 0;

If BullCx and MRO(BearCx,4,1) = -1 then
    Buy next bar at Close of this bar limit;

If BearCx and MRO(BullCx,4,1) = -1 then
    SellShort next bar at Close of this bar limit;

If Mom < Mom[1] and Mom[1] < Mom[2] then
    Sell next bar at market;

If Mom > Mom[1] and Mom[1] > Mom[2] then
    BuytoCover next bar at market;
```

Look again at the two entry statements and the use of the `MRO` function. In the first statement:

```
If BullCx and MRO(BearCx,4,1) = -1 then
    Buy next bar at Close of this bar limit;
```

generates a limit order to buy during the next bar (at the closing price of this bar) if:

- `BullCx` is true, meaning momentum crosses over 0 and
- `MRO` returns a value of `-1`, meaning there were no instances of a `BearCx` over the last 4 bars.

The second entry:

```
If BearCx and MRO(BullCx,4,1) = -1 then  
  SellShort next bar at Close of this bar limit;
```

generates a Limit order to sell short during the next bar (at the closing price of this bar) if:

- BearCx is true, meaning momentum crosses under 0 and
- MRO returns a value of -1, meaning there were no instances of a BullCx over the last 4 bars.

Verify what you have typed and place this strategy on a chart to view its performance. Insert the indicator **#11 Momentum**, as well. This will make it possible to see positions generated by momentum crossing over and under 0, as well as positions not taken due to recent prior crosses as identified by the MRO function.

*25 Momentum Cross



Challenge 6: *26 Key Reversal

Learning objective: Using user-declared true/false variables; describing bar patterns; limit order syntax.

Write a strategy that uses key reversals up and key reversals down to identify entry points.

Declare and assign variables for key reversals up and down.

Have the strategy enter a limit order to buy on the bar following a key reversal up, at a limit price better than the current bar's close.

Have the strategy enter a limit order to sell short on the bar following a key reversal down, at a limit price better than the current bar's close.

Declare an input for the number of points above or below the reversal bar's close to set the limit order prices; have the input default to 5 points.

Pre-declared True/False Variables

Just as with pre-declared numeric variables, which use `Value0` through `Value99`, one hundred names have been reserved for pre-declared true/false variables. These names are:

`Condition0` through `Condition99`.

These names are automatically recognized as true/false variables and simply require the assignment of a true/false expression to the name. This is done with a variable assignment statement:

Variable Assignment Statement

`Condition0 = true/false expression;`

To use the name `Condition1` to refer to the true/false test of the current close being greater than the previous close throughout an analysis technique or strategy:

`Condition1 = Close > Close[1];`

26. Built-in Stops

EasyLanguage includes built-in exit commands that may be included directly in your strategies. These special commands, which are reserved words, will be active even on the bar of entry; that is, they are evaluated on each tick (regardless of any setting for Intrabar Order Generation on the *Calculation* tab of the *Format Strategy* dialog) and not only on the close of the bar.

Although you may create any stop and exit formulas you choose, these built-in stops are convenient for adding risk and trade management to your strategies with just a few EasyLanguage words. An additional benefit is that these built-in stops include exits from both long and short positions in a single command.

These exits may be set to operate on a position basis or share/contract basis.

- `SetStopPosition` - exit is calculated for the entire position.
- `SetStopShare` or `SetStopContract` - exit is calculated per share or contract.

Note: By default, built-in stops are set on a position basis. You can specify built-in stop amounts on a per share/contract basis using `SetStopShare` or `SetStopContract`. You can view an example of this in the next exercise, **27 Breakout2*.

The commands listed below are used to enable a specific exit:

- `SetBreakEven` – sets an exit stop at the entry price, after a minimum profit is achieved.
- `SetDollarTrailing` – sets an exit stop a fixed number of dollars away from the peak profit.
- `SetPercentTrailing` - sets an exit stop a fixed percent of the peak profit away from the peak profit, after a minimum profit is achieved.
- `SetProfitTarget` – sets an exit order at a fixed dollar profit target.
- `SetStopLoss` – sets a stop loss order at a fixed dollar risk from entry.

Exercise: *27 Breakout2

Learning objective: Using built-in stops in custom strategies.

Description: This is a re-write of the previously written strategy, **23 Breakout*. The strategy buys (long) on a breakout above the highest high of the preceding 8 bars; it sells short on a break down below the lowest low of the preceding 8 bars. Built-in stops have been added.

In the TradeStation Development Environment, open the Editor window for the strategy *#23 Breakout*. Use the *File – Save As* menu sequence to save this as *#27 Breakout2*. As stated above, this is a re-write of that earlier exercise which appears below:

```
Input: Length(8);
Vars: BuyPx(0), SellPx(0);

BuyPx = Highest(High, Length) + .02;
SellPx = Lowest(Low, Length) - .02;

Buy ("Brk LE") next bar at BuyPx Stop;
Sell Short ("Brk SE") next bar at SellPx Stop;
```

The first change, adding `SetStopShare`, sets the exits to be used to operate on a per share/contract basis:

```
Input: Length(8);
Vars: BuyPx(0), SellPx(0);
BuyPx = Highest(High, Length) + .02;
SellPx = Lowest(Low, Length) - .02;
Buy ("Brk LE ") next bar at BuyPx Stop;
Sell Short ("Brk SE") next bar at SellPx Stop;

SetStopShare;
```

Next, the exit commands are added, including their respective inputs which are declared with default values of our choosing:

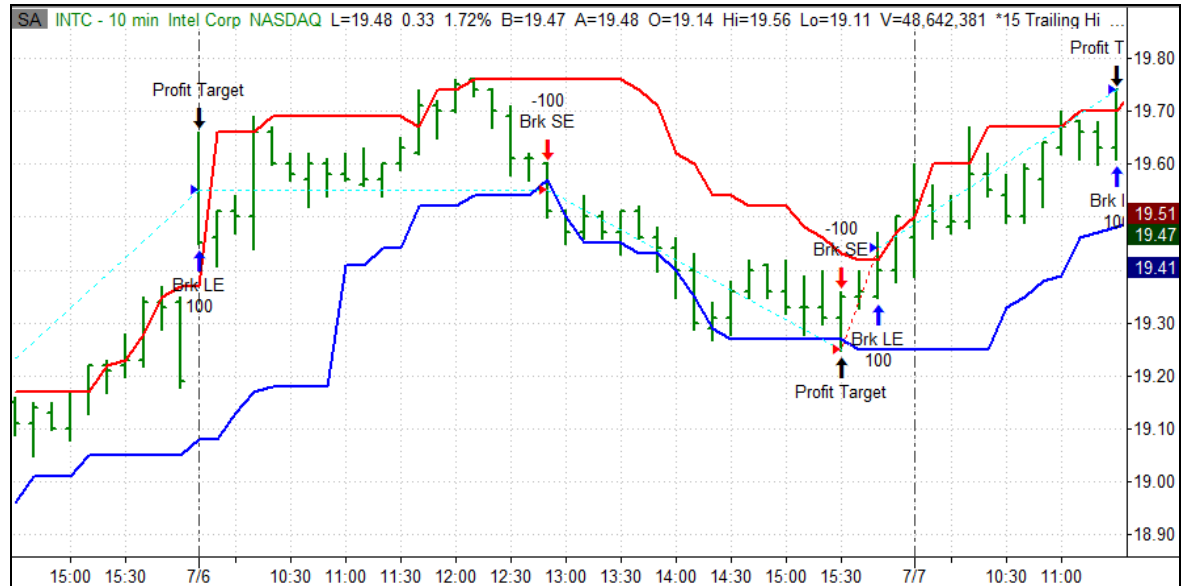
```
Input: Length(8), StopAmt(.2), TargetAmt(.3), BEAmt(.15);
Vars: BuyPx(0), SellPx(0);
BuyPx = Highest(High, Length) + .02;
SellPx = Lowest(Low, Length) - .02;
Buy ("Brk LE") next bar at BuyPx Stop;
Sell Short ("Brk SE") next bar at SellPx Stop;

SetStopShare;

SetStopLoss(StopAmt);
SetProfitTarget(TargetAmt);
SetBreakeven(BEAmt);
```


In this example, we have added a stop loss, profit target, and breakeven stop. Verify your work and place this strategy on a chart. The indicator **#15 Trailing Hi Lo** can be inserted into the chart as well.

***27 Breakout2**



Exercise: *28 RSI OB OS

Learning objective: Using Built-in Stops in custom Strategies; using **HighestBar** and **LowestBar** Functions.

Description: This strategy takes long (short) positions when the oversold (overbought) RSI begins to turn. For longs, RSI must be below 50 and a 7-bar low must have been made at least 3 bars ago. Shorts are the reverse. Built-in stops have been added.

Create a new strategy named *#28 RSI OB OS*. First declare and assign a variable for RSI. This function requires 2 parameters. Use *Close* for the *Price* parameter; the *Length* parameter will be an input.

```
Var: xRSI(0);  
  
xRSI = RSI( Close, Length );
```

Next add the entry rules using the **LowestBar** and **HighestBar** functions. These functions return the number of bars ago that the lowest (highest) value of the *Price* parameter occurred, over the last *Length* bars.

```
Var: xRSI(0);  
  
xRSI = RSI( Close, Length );  
  
If xRSI < 50 and LowestBar(xRSI, 7) >= 3 then  
    Buy next bar at market;  
  
If xRSI > 50 and HighestBar(xRSI, 7) >= 3 then  
    SellShort next bar at market;
```

Now use several built-in stops to manage the trades. The stop levels will be declared as inputs.

```
Var: xRSI(0);  
  
xRSI = RSI( Close, Length );  
  
If xRSI < 50 and LowestBar(xRSI, 7) >= 3 then  
    Buy next bar at market;  
  
If xRSI > 50 and HighestBar(xRSI, 7) >= 3 then  
    SellShort next bar at market;  
  
SetStopLoss(StopAmt);  
SetBreakeven(BEAmt);  
SetDollarTrailing(TrlgAmt);
```

Finally, add the input declaration statement for *Length* and the built-in stop levels.

```

Input: Length(14), StopAmt(50), BEAmt(50), TrlgAmt(100);

Var: xRSI(0);

xRSI = RSI( Close, Length );

If xRSI < 50 and LowestBar(xRSI, 7) >= 3 then
    Buy next bar at market;

If xRSI > 50 and HighestBar(xRSI, 7) >= 3 then
    SellShort next bar at market;

SetStopLoss(StopAmt);
SetBreakeven(BEAmt);
SetDollarTrailing(TrlgAmt);

```

Verify the strategy, and then return to a Chart Analysis window in the TradeStation Platform. Insert the indicator **RSI**. Change the values of both the OverSold and OverBought inputs to 50. Doing with this will have the indicator plots conform to the concept of the strategy, making it easier to examine on the chart. Now add the strategy to the chart.

#28 RSI OB_OS



27. Multidata: Referencing Multiple Data Sets

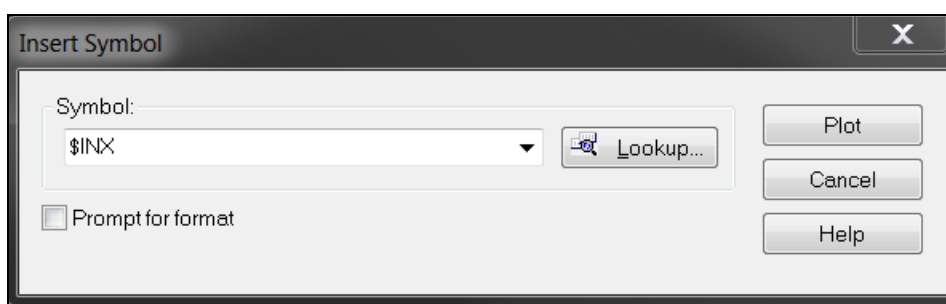
Each Chart Analysis window may contain up to 50 distinct data sets (symbols and time-based intervals). Each symbol is assigned a data number as it is added to the window, “Data1” through “Data50.” EasyLanguage for analysis techniques is always processed for Data1 unless specified otherwise. EasyLanguage for strategies, too, is always processed for Data1 unless otherwise specified; a strategy may buy and sell only Data1 even when other data sets are referenced for calculation purposes.

Exercise: *29 Multidata MA

Learning objective: Referencing multiple data sets in a chart.

Description: This strategy uses 2 data sets to generate long signals only. They are taken when the Close of Data1 is above its 20-bar average and the Close of Data2 is above its 50-bar average.

First, you'll need to create a chart with 2 symbols. To do this, create a chart as you normally would. In this example, create the chart using the symbol SPY. That will assign it as Data1 automatically. Then select *Insert - Symbol* from the main menu. Type a second symbol in the *Insert Symbol* dialog that appears and click *Plot*. Use \$INX for Data2.



Returning to the Development Environment, create a new strategy called **#29 Multidata MA**. Begin by declaring inputs and variables. The names will help us remember which data set we are referring to:

```
Inputs: AvgLenD1(20), AvgLenD2(50);
```

```
Variables: AvgD1(0), AvgD2(0);
```

Next, the variable assignment statements. In these statements, the references to Data1 and Data2 are not mere reminders: they are instructions to use the *Close* of the respective data sets in the calculations.

```
Inputs: AvgLenD1(20), AvgLenD2(50);
```

```
Variables: AvgD1(0), AvgD2(0);
```

```
AvgD1 = Average(Close Data1, AvgLenD1);
```

```
AvgD2 = Average(Close Data2, AvgLenD2);
```

AvgD1 is the 20-bar average of Data1, and AvgD2 is the 50-bar average of Data2.

Simple trading signals are added for long entry and exit only.

```
Inputs: AvgLenD1(20), AvgLenD2(50);  
Variables: AvgD1(0), AvgD2(0);
```

```
AvgD1 = Average(Close Data1, AvgLenD1);  
AvgD2 = Average(Close Data2, AvgLenD2);
```

```
If Close > AvgD1 and Close Data2 > AvgD2 then  
    Buy next bar at market  
else  
    Sell next bar at market;
```

Return to the multidata chart you created in the TradeStation Platform. Insert the indicator **Mov Avg 1 Line** two times.

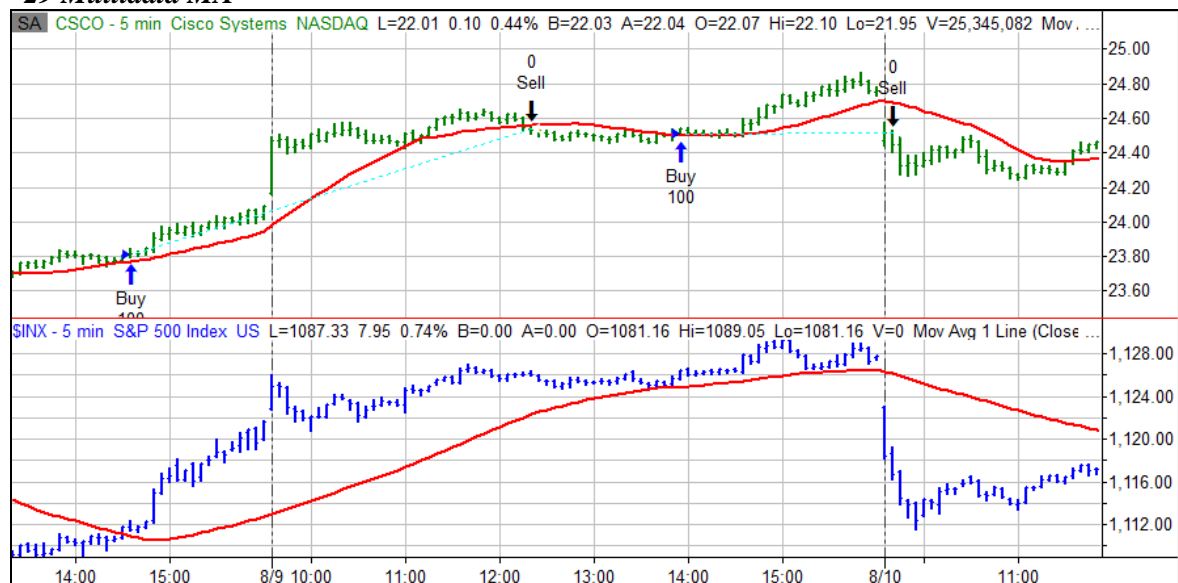
In the first instance, use the *Insert – Indicator* menu sequence, select **Mov Avg 1 Line** and be sure *Prompt for Format* is checked. On the *Inputs* tab, set the *Length* input to 20.

In the second instance, use the *Insert – Indicator* menu sequence, select **Mov Avg 1 Line** and be sure *Prompt for Format* is checked. On the *Inputs* tab, set the *Length* input to 50. On the *General* tab, in the *Base study on:* dropdown, select 2: \$INX.

These steps will calculate and plot the two moving averages on the correct data sets in the chart.

Now insert the new strategy into the chart.

*29 Multidata MA



28. Dynamic Formatting

Plot color, plot line width and cell background color may be set within an Analysis Technique using the Reserved Words *SetPlotColor*, *SetPlotWidth* and *SetPlotBGColor*.

Charting and RadarScreen:	<code>SetPlotColor(<i>PlotNumber</i>, <i>Color</i>);</code>
Charting:	<code>SetPlotWidth(<i>PlotNumber</i>, <i>Width</i>);</code>
RadarScreen:	<code>SetPlotBGColor(<i>PlotNumber</i>, <i>Color</i>);</code>

For all 3 of these commands, *PlotNumber* is a number from 1 to 99 representing the number of the plot to modify.

The parameter *Color* is the EasyLanguage color to be used for the plot. The following 16 color names are available:

Black
Blue
Cyan
Green
Magenta
Red
Yellow
White
DarkBlue
DarkCyan
DarkGreen
DarkMagenta
DarkRed
DarkBrown
DarkGray
LightGray

For example,

```
SetPlotColor(1, yellow);
```

Note: These 16 colors may also be identified by number. In addition, a full palette of colors may also be identified by their RGB (red, green, blue) values. For more information, see the Dictionary and the Help in the TDE for EasyLanguage Reserved Words and Functions.

The parameter *Width* is the EasyLanguage width to be used for the plot. You can specify the width of a plot by using any numeric value from 0 through 6, where 0 represents the narrowest width and 6 the thickest. These statements may be included in *If...then* statements so that the plot color and width will change automatically according to the user-defined criteria in the true/false expression.

Exercise: *30 High Volume Bars

Learning objective: Dynamic formatting using `SetPlotColor` and `SetPlotWidth` commands.

Description: This indicator plots a volume histogram, with the histogram bolder and in a different color on high volume bars.

Create a new indicator called **#30 High Volume Bars**. Declare inputs for the volume considered high and for the color and width of the volume histogram on those high volume bars.

```
Inputs: HiVolume(500000), LineWidth(4), LineColor(yellow);
```

We can use a block `If...then` for the color and width conditions.

```
Inputs: HiVolume(500000), LineWidth(4), LineColor(yellow);
```

```
If Ticks > HiVolume then begin
    SetPlotColor(1, LineColor);
    SetPlotWidth(1, LineWidth);
end;
```

Then add the plot statement.

```
Inputs: HiVolume(500000), LineWidth(4), LineColor(yellow);
If Ticks > HiVolume then begin
    SetPlotColor(1, LineColor);
    SetPlotWidth(1, LineWidth);
end;
```

```
Plot1(Ticks, "Volume");
```

Verify the indicator.

Insert **#30 High Volume Bars** into a chart. You may want to change the value of the `HiVolume` input depending on the symbol and interval you are charting.

#30 High Volume Bars



29. Color Gradients

The color of a plot (or the foreground or background in a grid such as RadarScreen) may be set along a color gradient based on the value of a variable. This allows even greater use of color as a visual cue to an indicator or price event.

```
GradientColor(Value1, -10, 10, Red, Green);
```

This allows TradeStation to select a color based on the value of Value1. A value of Value1 from -10 to 10 will generate a color along a gradient from red to green.

The next exercise uses a color gradient in an indicator designed for RadarScreen.

30. Plot Statements for RadarScreen

In a RadarScreen window, plot statements instruct TradeStation as to what information is to be placed in a RadarScreen cell. This information may be a numeric value, a true/false value, or text. Each plot statement creates a column for that analysis technique in the RadarScreen window.

```
PlotN(cell content, "plot name") ;
```

N = 1 to 99

Cell content is either a

- Numeric value to be displayed in the cell
- A true/false value to be displayed in the cell
- Text to be placed in the cell.

"plot name" is the label that will be used for this plot in the RadarScreen window's column headings and in the formatting dialogs.

Sample plot statements for RadarScreen:

This will place the value of `Condition1` (true or false) in the cell:

```
Plot1(Condition1, "MyCondition") ;
```

This will place the value of `Value1` (numeric value) in the cell:

```
Plot1(Value1, "MyCalculation") ;
```

This will place the text "UpBar" or "DownBar" in the cell:

```
If Close > Close[1] then  
    Plot1("UpBar", "Change")  
else  
    Plot1("DownBar", "Change") ;
```

Note: including *"plot name"* in a plot statement is optional, but recommended. Other optional parameters for formatting are also available.

Exercise: *31 Real Body Avg RS

Learning objective: Writing an indicator for RadarScreen.

Description: This indicator for RadarScreen places the word **Bullish** or **Bearish** in a cell based on the value of the average real body, and also displays the value of the average real body. The background of the cells is colored along a gradient based on the value of the average real body.

Create a new indicator named *#31 Real Body Avg RS*. The basic calculations are the same as those in *#12 Real Body Avg*, with other statements added to make this suitable for RadarScreen. An input is also added for this version, since we had not yet covered that topic when writing *#12 Real Body Avg*.

```
Input: Length(10);

Vars: RB(0), AvgRB(0);

RB = Close - Open;
AvgRB = Average(RB, Length);
```

Add a third variable assignment statement, a pre-declared true/false variable, *Condition1*.

```
Input: Length(10);
Vars: RB(0), AvgRB(0);

RB = Close - Open;
AvgRB = Average(RB, Length);
Condition1 = AvgRB > 0;
```

Now add the following *If...then...else* statement, which assigns text to a text variable which we will call *Direction*, based on the value of *AvgRB*. Also, add two plot statements which place the text and numeric variable values in columns in RadarScreen. Finally, you'll add two *SetPlotBGColor* statements to adjust the background color of the cells along a gradient based on the value of *AvgRB*.

```
Input: Length(10);

Vars: RB(0), AvgRB(0);

RB = Close - Open;
AvgRB = Average(RB, Length);
Condition1 = AvgRB > 0;

If Condition1 then
    Direction = "Bullish"
else
    Direction = "Bearish";
```

```

Plot1(Direction, "RB Dir");
Plot2(AvgRB, "AvgRB");

SetPlotBGColor(1, GradientColor(AvgRB, -.015, .015, red,
green));
SetPlotBGColor(2, GradientColor(AvgRB, -.015, .015, red,
green));

```

Let's examine what you have typed so far. You have created two plot statements, each of which will appear as a separate column in a RadarScreen window. The first plot, named `RB Dir`, will plot the word "Bullish" in the RadarScreen cell if the value of `AvgRB` is greater than zero or plot the word "Bearish" if the value of `AvgRB` is less than zero. The second plot, named `AvgRB` will plot the actual value of `AvgRB` in the RadarScreen cell.

In addition, the background colors of the cells for both plots will be red for negative values and green for positive values and scaled along a gradient based on the value of `AvgRB`. In other words, red and green shades will vary based on how bullish or bearish the value of `AvgRB` is.

Notice here that there is another variable, `Direction`. This variable needs to be declared in the variable declaration statement as follows: `Direction ("")`. This lets TradeStation know the variable `Direction` now exists and is a text variable. This is indicated by the double quotation marks in the declaration statement.

`Direction` is then assigned within the conditional `If...then...else` statement. This is done because the variable `Direction` is assigned a different value based on whether `Condition1` is `TRUE` or `FALSE`. In other words, if `Condition1` is true then `Direction = "Bullish"`. If `Condition1` is false, then `Direction = "Bearish"`.

```

Input: Length(10);

Vars: RB(0), AvgRB(0), Direction("");

RB = Close - Open;
AvgRB = Average(RB, Length);
Condition1 = AvgRB > 0;

If Condition1 then
    Direction = "Bullish"
else
    Direction = "Bearish";

Plot1(Direction, "RB Dir");
Plot2(AvgRB, "AvgRB");

SetPlotBGColor(1, GradientColor(AvgRB, -.015, .015, red, green));
SetPlotBGColor(2, GradientColor(AvgRB, -.015, .015, red, green));

```

Verify the indicator and right click in the Editor and select *Properties* to open the *Indicator Properties* dialog.

On the *General* tab, there is a checkbox and edit box labeled *Load additional data for accumulative calculations*. This setting is used only for studies to be applied to RadarScreen. It tells RadarScreen to load more data than it might otherwise think necessary so you can be sure it has adequate data to make your calculations. Click the *Details* button for more information on this setting. For this indicator, check this box and set the *Additional bars to load* to 20.

Click on the Grid Style tab. Set the plot for *AvgRB* to *Number (Category)* with 3 decimals. This will allow display of more detail in the value of *AvgRB*.

Click OK, verify the indicator again, and insert it into a RadarScreen window. You can sort by the numeric value column for the indicator to more easily see the color gradient.

Scaling		Applications		Alerts	
General	Chart Style	Grid Style	Chart Color	Grid Color	
Name:		Short name:			
<input type="text" value="31 Real Body Avg RS"/>		<input type="text" value="Filter RB"/>			
Notes:		<input type="text" value="EasyLanguage Boot Camp"/>			
Maximum number of bars study will reference:					
<input checked="" type="radio"/> Auto-detect <input type="radio"/> User defined: <input type="text" value="50"/>					
<input type="checkbox"/> Update value intra-bar (tick-by-tick) Note: This will allow alerts to be triggered intra-bar					
<input checked="" type="checkbox"/> Load additional data for accumulative calculations Additional bars to load: <input type="text" value="20"/>				<input type="button" value="Details..."/>	
Currency Based on:					
<input type="text" value="Symbol"/>					
Amount currency is set to USD by default in the case of no account specified.					
<input type="button" value="Default"/>					

***31 Real Body Avg RS**

	Symbol	Interval	*31 Real Body Avg RS		Last	Net Chg	Net %Chg ...	Bid
			RB Dir	AvgRB				
37	PFE	Daily	Bullish	0.073	16.33	-0.09	-0.55%	16.33
38	AES	Daily	Bullish	0.070	11.12	0.05	0.45%	11.12
39	FE	Daily	Bullish	0.055	37.20	-0.27	-0.72%	37.20
40	CNP	Daily	Bullish	0.054	14.87	-0.09	-0.59%	14.87
41	T	Daily	Bullish	0.054	26.83	-0.03	-0.11%	26.82
42	MRK	Daily	Bullish	0.050	35.31	-0.05	-0.14%	35.31
43	CNW	Daily	Bullish	0.047	29.86	-0.72	-2.35%	29.85
44	GE	Daily	Bullish	0.046	16.11	-0.27	-1.65%	16.11
45	AA	Daily	Bullish	0.037	11.20	-0.46	-3.95%	11.20
46	KFT	Daily	Bullish	0.034	30.25	-0.14	-0.46%	30.24
47	NI	Daily	Bullish	0.034	16.90	-0.10	-0.59%	16.90
48	LSTR	Daily	Bullish	0.015	39.91	-0.64	-1.58%	39.91
49	PG	Daily	Bullish	0.014	60.73	0.35	0.58%	60.73
50	JBLU	Daily	Bullish	0.011	6.39	-0.12	-1.77%	6.38
51	DUK	Daily	Bullish	0.007	17.60	0.04	0.23%	17.59
52	MSFT	Daily	Bullish	0.005	25.08	-0.53	-2.07%	25.07
53	DAL	Daily	Bullish	0.001	11.79	-0.40	-3.28%	11.79
54	LUV	Daily	Bearish	0.000	11.80	-0.34	-2.80%	11.79
55	AMR	Daily	Bearish	(0.012)	7.15	-0.16	-2.19%	7.15
56	CSX	Daily	Bearish	(0.020)	52.73	-1.14	-2.12%	52.69
57	TRV	Daily	Bearish	(0.032)	49.79	-0.82	-1.62%	49.78
58	JBHT	Daily	Bearish	(0.044)	34.91	-0.73	-2.05%	34.89
59	JPM	Daily	Bearish	(0.053)	39.32	-0.50	-1.26%	39.32
60	HD	Daily	Bearish	(0.057)	28.17	-0.53	-1.85%	28.16
61	BAC	Daily	Bearish	(0.072)	13.70	-0.21	-1.53%	13.69
62	INTC	Daily	Bearish	(0.082)	19.87	-0.78	-3.78%	19.86
63	AXP	Daily	Bearish	(0.093)	43.44	-0.32	-0.72%	43.43
64	PEG	Daily	Bearish	(0.108)	32.34	-0.33	-1.01%	32.34
65	HPQ	Daily	Bearish	(0.154)	42.60	0.00	0.00%	42.60
66	CAL	Daily	Bearish	(0.163)	23.75	-0.76	-3.10%	23.75

Page 1

31. Variables: Capturing Values

Variables hold their value from bar to bar until they are re-assigned, i.e., reset or recalculated. This allows the use of variables to capture a value and hold it indefinitely for further use. When doing this, the variable assignment statement is generally placed inside a conditional so that the variable is set or calculated only when the condition is true. The variable holds the new value until the next time the condition is true.

For example:

```
If Time = SessionFirstBarTime(1, 1) then  
    Value1 = Close - Close[1];
```

This will calculate the net change from the previous day's close to the current day's first-bar close. On an intraday chart, the calculation is made only on the first bar of each day; `Value1` then retains that value until the first bar of the next day.

Now try an exercise which uses variables to capture and retain values.

Exercise: *32 Intraday H and L

Learning objective: Using variables to capture and retain values; using Session references.

Description: This indicator plots 2 lines: the high and low for the day (session) through the current bar. This indicator is intended for use on intraday charts.

Create a new indicator named *#38 Intraday H and L*. In the Editor, declare two numeric variables *DHigh* and *DLow* and use these variables in an *If...then* statement to capture and hold the High and Low of the first bar of the day. The first bar of the day, or session, is identified by comparing the time stamp of the bar to the time returned by the function *SessionFirstBarTime*. If they are equal, we are at the first bar of the day.

```
Var: DHigh(0), DLow(0);

If Time = SessionFirstBarTime(1, 1) then begin
    DHigh = High;
    DLow = Low;
end
```

Next, we need to add the EasyLanguage to check for new highs and lows after the first bar of the day.

```
Var: DHigh(0), DLow(0);

If Time = SessionFirstBarTime(1, 1) then begin
    DHigh = High;
    DLow = Low;
end
else begin
    If High > DHigh then
        DHigh = High;
    If Low < DLow then
        DLow = Low;
end;
```

Finally, two plot statements are added within an If...then statement to plot the values of DHigh and DLow. This is done to ensure consistency in scaling throughout the entire range of the chart.

```
Var: DHigh(0), DLow(0);

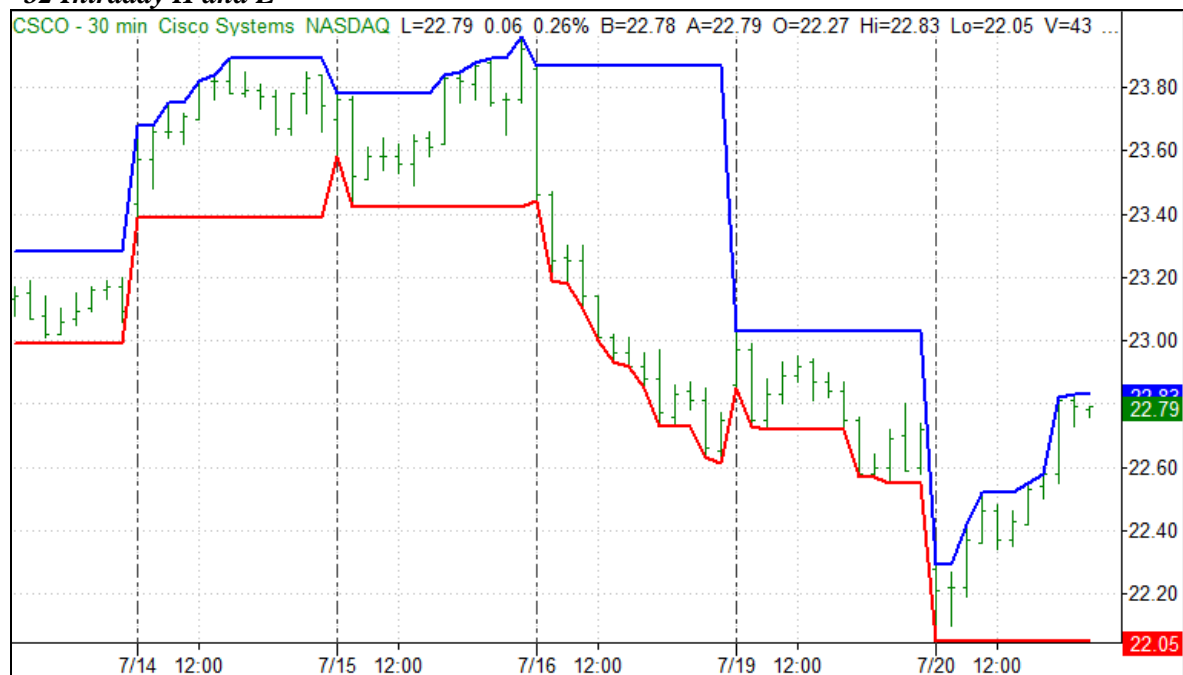
If Time = SessionFirstBarTime(1, 1) then begin
    DHigh = High;
    DLow = Low;
end
else begin
    If High > DHigh then
        DHigh = High;
    If Low < DLow then
        DLow = Low;
end;

If DLow <> 0 then begin
    Plot1(DHigh, "DayHigh");
    Plot2(DLow, "DayLow");
end;
```

Verify the EasyLanguage and set the Indicator *Properties – Scaling to Same Axis as Underlying Data*.

Insert this indicator on a chart. Notice highs and lows are captured and plotted for the first bar of the day. Those values are held until new highs or lows for the day are achieved, which then become the new plotted values.

*32 Intraday H and L



32. Variables: Counting Events

Variables may also be used as counters. Placing a variable inside a conditional allows use of the variable for counting events. The variable may then be incremented (or decremented) only when the condition is true.

Because variables retain their value until recalculated, at times it may be necessary to reset the variable, such as when the condition becomes false.

```
Vars: UpCount(0);  
  
If Close > Close[1] then  
    UpCount = UpCount + 1  
else  
    UpCount = 0;
```

The variable `UpCount` tracks the number of consecutive higher closes. It is reset to 0 whenever there is not a higher close.

Exercise: *33 Streak

Learning objective: Using variables to count events; resetting variables.

Description: This indicator displays the current streak of up bars or down bars; in other words, the current number of consecutive up or down bars. It is intended for use in RadarScreen.

Create a new indicator named *#33 Streak*. First, declare two variables, UpCount and DownCount. UpCount will be used to count the number of consecutive up bars. DownCount will be used to count the number of consecutive down bars.

```
Var: UpCount(0), DownCount(0);
```

Next add EasyLanguage that will count consecutive higher closes or reset the counter to zero when there is not a higher close. Of course you'll add EasyLanguage for the reverse of this as well, which will maintain the count of consecutive lower closes or reset the counter to zero when there is not a lower close. Include an input for added flexibility.

```
Inputs: Price(Close);
Var: UpCount(0), DownCount(0);

If Price > Price[1] then
    UpCount = UpCount + 1
else
    UpCount = 0;

If Price < Price[1] then
    DownCount = DownCount - 1
else
    DownCount = 0;
```

Notice that DownCount is decremented so that the number of consecutive lower closes is expressed as a negative number. This will make it easier to see and sort the column in RadarScreen.

Plot-background color specifications for UpColor and DownColor are added as inputs. And the Plot and Color commands are placed inside an If...then...else so that the variable plotted and the background color of the cell will be contingent on the current streak.

```
Inputs: Price(Close), UpColor(green), DnColor(red);
Var: UpCount(0), DownCount(0);

If Price > Price[1] then
    UpCount = UpCount + 1
else
    UpCount = 0;

If Price < Price[1] then
```



```

        DownCount = DownCount - 1
    else
        DownCount = 0;

    If UpCount > 0 then begin
        Plot1(UpCount, "Streak");
        SetPlotBGColor(1, UpColor);
    end
    else begin
        Plot1(DownCount, "Streak");
        SetPlotBGColor(1, DnColor);
    end;
end;

```

Looking at this from a trader's point of view, there's one other item that should be added to address a condition that has not yet been accounted for. What about an unchanged bar? Of course, it will ultimately be up to you to choose how to account for an unchanged bar. In this case, we decided to treat an unchanged bar as continuing the current streak. Let's add the EasyLanguage for this.

```

Inputs: Price(Close), UpColor(green), DnColor(red);

Var: UpCount(0), DownCount(0);

If Price > Price[1] or (UpCount > 0 and Price = Price[1])
then
    UpCount = UpCount + 1
else
    UpCount = 0;

If Price < Price[1] or (DownCount < 0 and Price = Price[1])
then
    DownCount = DownCount - 1
else
    DownCount = 0;

If UpCount > 0 then begin
    Plot1(UpCount, "Streak");
    SetPlotBGColor(1, UpColor);
end
else begin
    Plot1(DownCount, "Streak");
    SetPlotBGColor(1, DnColor);
end;
end;

```

Verify the indicator and right-click in the Editor and select *Properties* to open the *Indicator Properties* dialog. On the *General* tab, there is a checkbox and edit box labeled *Load additional data for accumulative calculations*. This setting is used only for studies to be applied to RadarScreen. It tells RadarScreen to load more data than it might otherwise think necessary so you can be sure it has adequate data to make your calculations. Click the *Details* button for more information on this setting. For this indicator, check this box and set the *Additional bars to load* to 20. Re-verify the indicator and insert it into a RadarScreen window.

You may be wondering what would happen if you used this indicator in a chart. Feel free to experiment and insert it into a chart. As written, it will create an oscillator from the UpCount and DownCount values.

However, here are 2 suggested enhancements to make it more useful in a chart. First, add commands to *SetPlotColor* to the same blocks with *SetPlotBGColor*. This will enhance the appearance of the indicator in the chart. Second, in the Indicator Properties dialog, Chart Style tab set the plot to Histogram.

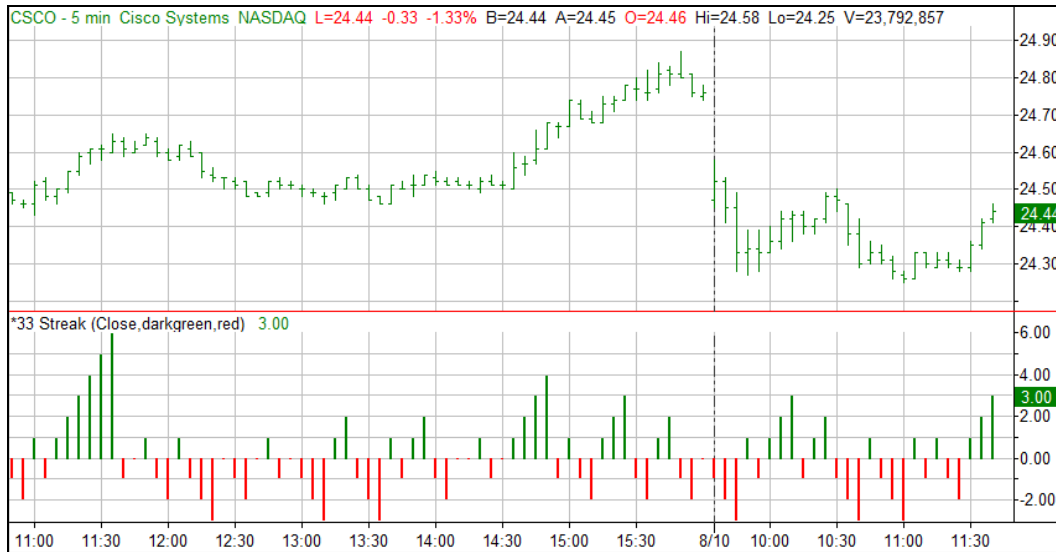
```
Inputs: Price(Close), UpColor(green), DnColor(red);
Var: UpCount(0), DownCount(0);

If Price > Price[1] or (UpCount > 0 and Price = Price[1]) then
    UpCount = UpCount + 1
else
    UpCount = 0;
If Price < Price[1] or (DownCount < 0 and Price = Price[1])
then
    DownCount = DownCount - 1
else
    DownCount = 0;
If UpCount > 0 then begin
    Plot1(UpCount, "Streak");
    SetPlotColor(1, UpColor);
    SetPlotBGColor(1, UpColor);
end
else begin
    Plot1(DownCount, "Streak");
    SetPlotColor(1, DnColor);
    SetPlotBGColor(1, DnColor);
end;
```

***33 Streak**

	Sym...	Interval	*33 Streak	Last	Net Chg	Net %Chg ...	Bid
1	AXP	Daily	2	41.85	0.27	0.65%	41.84
2	CAT	Daily	2	66.39	1.59	2.45%	66.38
3	CSCO	Daily	1	22.86	0.13	0.57%	22.85
4	CVX	Daily	2	72.75	0.75	1.04%	72.73
5	DD	Daily	2	36.54	0.49	1.36%	36.54
6	DIS	Daily	-1	33.18	-0.13	-0.39%	33.18
7	GE	Daily	2	14.93	0.31	2.13%	14.93
8	HPQ	Daily	-1	46.55	-0.13	-0.28%	46.55
9	IBM	Daily	-1	126.04	-3.75	-2.89%	126.02
10	INTC	Daily	2	21.59	0.00	0.01%	21.59
11	JPM	Daily	2	39.27	0.23	0.59%	39.27
12	KFT	Daily	2	29.10	0.27	0.94%	29.09
13	KO	Daily	1	53.04	0.77	1.47%	53.03
14	MCD	Daily	1	70.49	0.58	0.83%	70.48
15	MMM	Daily	2	81.96	0.74	0.91%	81.95
16	MRK	Daily	-3	35.50	-0.30	-0.84%	35.49
17	MSFT	Daily	2	25.30	0.07	0.28%	25.29
18	PFE	Daily	-1	14.57	-0.16	-1.09%	14.57
19	PG	Daily	1	61.88	0.02	0.03%	61.88
20	T	Daily	-1	24.69	-0.19	-0.76%	24.68
21	TRV	Daily	2	49.93	0.22	0.44%	49.92
22	UTX	Daily	2	67.27	0.88	1.33%	67.26
23	XOM	Daily	2	58.83	0.40	0.68%	58.83

***33 Streak**



Exercise: *34 Trade Highs Lows

Learning objective: Using variables to count events; using built-in stops.

Description: This strategy enters a limit order to buy following the number of consecutive new lows as determined by the input `LowsToBuy`. The strategy enters a limit order to sell short following the number of consecutive new highs as determined by the input `HighsToSell`. The limit price is set to the close of the signal bar +/- the number of points set in the input `Pts`. Profit target and stop loss orders are used on a position basis with the values in the inputs `ProfitTgt` and `StopLoss`, respectively.

Create a new strategy named *#34 Trade_Highs Lows*. As described above, this strategy will place a limit order to buy following a number of consecutive new lows, and place a limit order to sell following a number of consecutive new highs. From a trading perspective, a strategy such as this might be considered contrarian in nature. In other words, if a few lower lows are made, the strategy is going to buy into the move. Conversely, if a few higher highs are achieved, the strategy is going to sell into the move.

The first step is to declare two variables, call them `HiCount` and `LoCount`, and then use each one in a separate `If...then...else` statement to count higher highs and lower lows.

```
Var: HiCount(0), LoCount(0);
```

```
If Low < Low[1] then  
    LoCount = LoCount + 1  
else  
    LoCount = 0;
```

```
If High > High[1] then  
    HiCount = HiCount + 1  
else  
    HiCount = 0;
```

Next, you'll want to add the order entry portion of the strategy which will result in the generation of Limit orders to:

1. Buy after a certain number of consecutively occurring lower lows
2. Sell short after a certain number of consecutively occurring higher highs

How many lower lows or higher highs must occur before an order is generated is up to you. However, it would be a good idea to use an input for this, and in this exercise, `LowsToBuy` and `HighsToSell` are used; both defaulting to a value of two. In addition, as per the exercise instructions, limit prices are set to the closing price of the signal bar +/- the number of points set in the input `Pts`, which in this exercise is set to .02. Finally, profit target and stop loss orders are used on a **position basis** with the values in the inputs `ProfitTgt` and `StopLoss`, respectively.

In this exercise, these values are each set to \$50 for the overall size of the position. The completed strategy should look like as below.

```
Input: LowsToBuy(2), HighsToSell(2), Pts(.02), ProfTgt(50),
StopLoss(50);
```

```
Var: HiCount(0), LoCount(0);
If Low < Low[1] then
    LoCount = LoCount + 1
else
    LoCount = 0;
If High > High[1] then
    HiCount = HiCount + 1
else
    HiCount = 0;
```

```
If LoCount >= LowsToBuy then
    Buy next bar at close of this bar - Pts limit;
```

```
If HiCount >= HighsToSell then
    SellShort next bar at close of this bar + Pts limit;
```

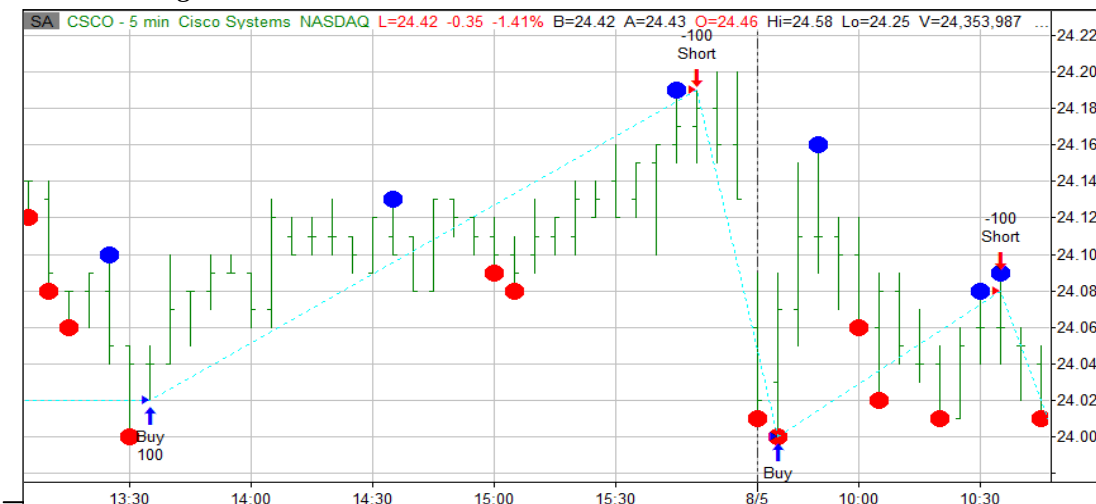
```
SetStopPosition;
SetProfitTarget(ProfTgt);
SetStopLoss(StopLoss);
```

Verify the strategy and insert it into a chart. You may also want to insert the *ShowMe* studies *Consecutive Downs* and *Consecutive Ups*. These will provide a visual aid to seeing the signal setups.

Format *Consecutive Downs* with the input *Price* set to *Low* and the input *ConsecutiveBarsDown* set to 2, or other value equaling the input *LowsToBuy* in the strategy.

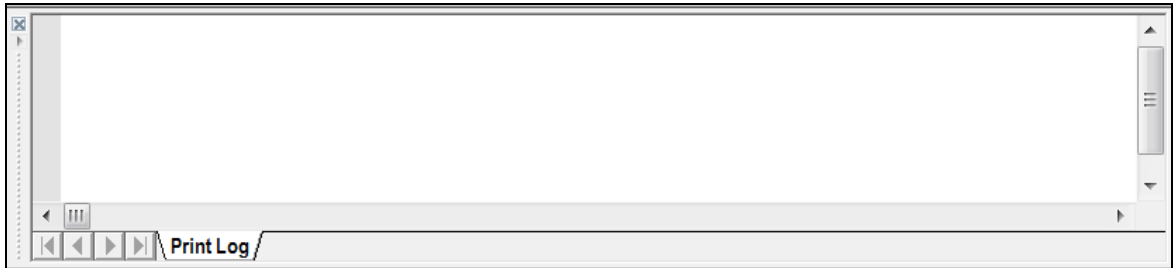
Format *Consecutive Ups* with the input *Price* set to *High* and the input *ConsecutiveBarsUp* set to 2, or other value equaling the input *HighsToSell* in the strategy.

*34 Trade_Highs Lows



33. Viewing Output from EasyLanguage: Print Log

The TradeStation Platform desktop can display the *EasyLanguage Print Log*. You may view the *Print Log* by using the *View – EasyLanguage Print Log* menu sequence in the TradeStation Platform (not the Development Environment). By default, the *Print Log* appears as a bar across the bottom of the desktop.




Instructions may be written into an analysis technique or strategy to send text, numeric values and true/false values to the *Print Log*. The proper format for a print statement is indicated in the example below. Items in quotes are text, such as labels for the values that follow them.

```
Print("Date",Date,"Value1",Value1,"Condition1",Condition1);
```

This command sends to the *Print Log* the Date, the value of Value1 and the value (true or false) of Condition1 for each bar. Each value is preceded by a label in quotes.

34. Viewing Output from EasyLanguage: Analysis Commentary

Instructions may also be written into an analysis technique or strategy to send text, numeric values and true/false values to the *Analysis Commentary* window. Analysis Commentary windows may be opened only from Chart Analysis windows (as well as RadarScreen or OptionStation windows) to which an analysis technique or strategy containing such instructions has been applied. To view an *Analysis Commentary* window, you may use the  button on the toolbar, or the *View – Analysis Commentary* menu sequence, and then click on the bar on the chart for which you wish to see *Analysis Commentary*.

The proper format for a commentary statement is written below:

```
Commentary("Date",Date,"Value1",Value1,"Condition1",Condition1);
```

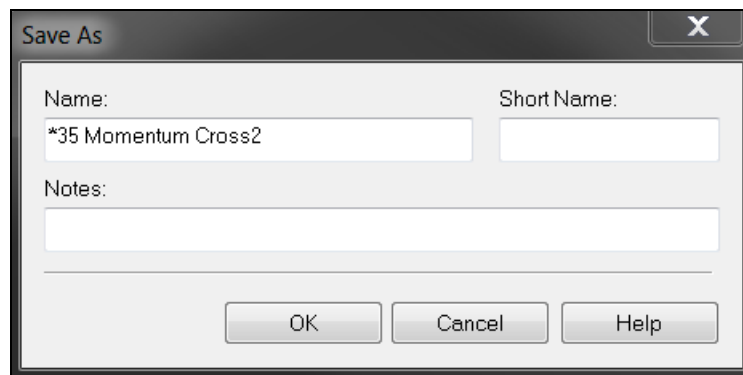
This command sends to the Analysis Commentary window the date, the value of Value1 and the value (true or false) of Condition1 for each bar. Each value is preceded by a label in quotes.

Exercise: *35 Momentum Cross2

Learning objective: Using **Print** and **Commentary** statements to send information to the **Print Log** and **Analysis Commentary**.

Description: This is a rewrite of **25 Momentum Cross*. This strategy buys (long) when momentum crosses over 0, as long as it has not crossed under 0 within the last 4 bars; it sells short when momentum crosses under 0, as long as it has not crossed over 0 within the last 4 bars; it sells (liquidates long positions) if momentum declines on 2 consecutive bars; it buys (covers short positions) if momentum rises on 2 consecutive bars.

In the Development Environment, open or access **25 Momentum Cross* and from the main menu select *File* then *Save As...* You will see the dialog below which will allow you to create an additional version of this strategy with a different name. Rename the document *#35 Momentum Cross2* and click *OK*.



This saves an additional copy of the strategy with a new name. You will now add both print and commentary statements to the existing strategy.

First add the print statement to the end of the existing EasyLanguage:

```
Print("Date", Spaces(2), Date:7:0, Spaces(2), "Time",  
Spaces(2), Time:4:0, Spaces(2), "Mom", Spaces(2), Mom,  
Spaces(2), "BullCr", Spaces(2), BullCr, Spaces(2), "BearCr",  
Spaces(2), BearCr);
```

Now add the commentary statement to send information to the Analysis Commentary window as well:

```
Commentary("Date", Spaces(2), Date:7:0, Spaces(2), "Time",  
Spaces(2), Time:4:0, Spaces(2), "Mom", Spaces(2), Mom,  
Spaces(2), "BullCr", Spaces(2), BullCr, Spaces(2), "BearCr",  
Spaces(2), BearCr);
```

Take a closer look at the information contained in both the print and commentary statements:

- "Date" - The word *Date* contained in quotes is a label for the date value to follow.
- Spaces – A reserved word that will add spaces between other labels and values.
- Date:7:0 - In EasyLanguage, the reserved word date returns a numeric expression representing the EasyLanguage date of the closing price of the bar being analyzed. This numeric expression is in the form YYYYMMDD, where YYYY is years since 1900, MM is the month, and DD is the day of the month. For example, *1030611* represents June 11, 2003. This EasyLanguage `Date:7:0` instructs TradeStation to report the date as seven digits to the left of the decimal point and zero digits after the decimal point.
- "Time" - The word *Time* contained in quotes is the text label for *Time*.
- Time:4:0 - This reserved word returns a numeric expression representing the EasyLanguage time (HHMM format) of the closing price of the current bar. For example, 1600 is returned if the time of the bar is 4:00 PM. `Time:4:0` instructs TradeStation to report the time as four digits (in military time) to the left of the decimal point and zero after it.
- "Mom" - This is the text label for the value of *Mom*.
- Mom - The value for *Mom*. From the EasyLanguage in the strategy, *Mom* is equal to the Momentum: `Mom = Momentum(Close, Length)`.
- "BullCx" - This is the text label for *BullCx*.
- BullCx - The value for *BullCx*. This will be reported as either TRUE or FALSE.
- "BearCx" - This is the text label for *BearCx*.
- BearCx - The value for *BearCx*. This will be reported as either TRUE or FALSE.

Your finished strategy should look as follows:

```
Input: BarsAgo(10);

Vars: Mom(0), BullCx(false), BearCx(false);

Mom = Momentum(Close,BarsAgo);
BullCx = Mom crosses over 0;
BearCx = Mom crosses under 0;

If BullCx and MRO(BearCx,4,1) = -1 then
    Buy next bar at Close of this bar limit;

If BearCx and MRO(BullCx,4,1) = -1 then
    SellShort next bar at Close of this bar limit;

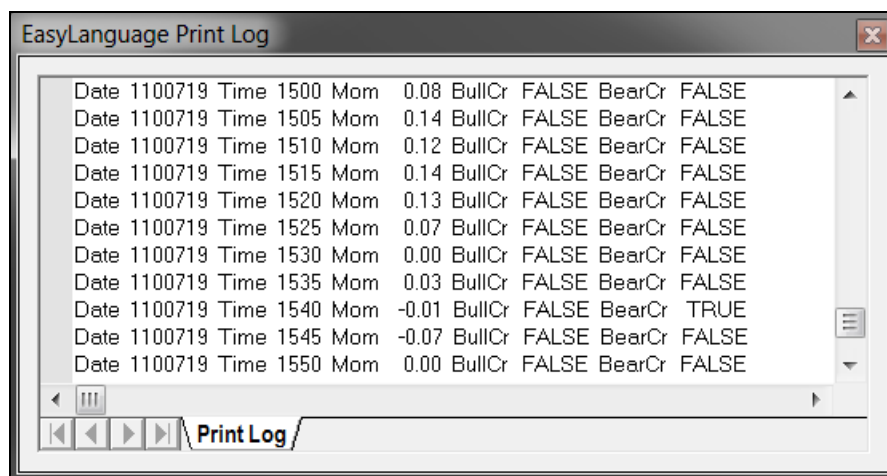
If Mom < Mom[1] and Mom[1] < Mom[2] then
    Sell next bar at market;


If Mom > Mom[1] and Mom[1] > Mom[2] then
    BuyToCover next bar at market;

Print("Date", Spaces(2), Date:7:0, Spaces(2), "Time",
Spaces(2), Time:4:0, Spaces(2), "Mom", Spaces(2), Mom,
Spaces(2), "BullCr", Spaces(2), BullCr, Spaces(2), "BearCr",
Spaces(2), BearCr);

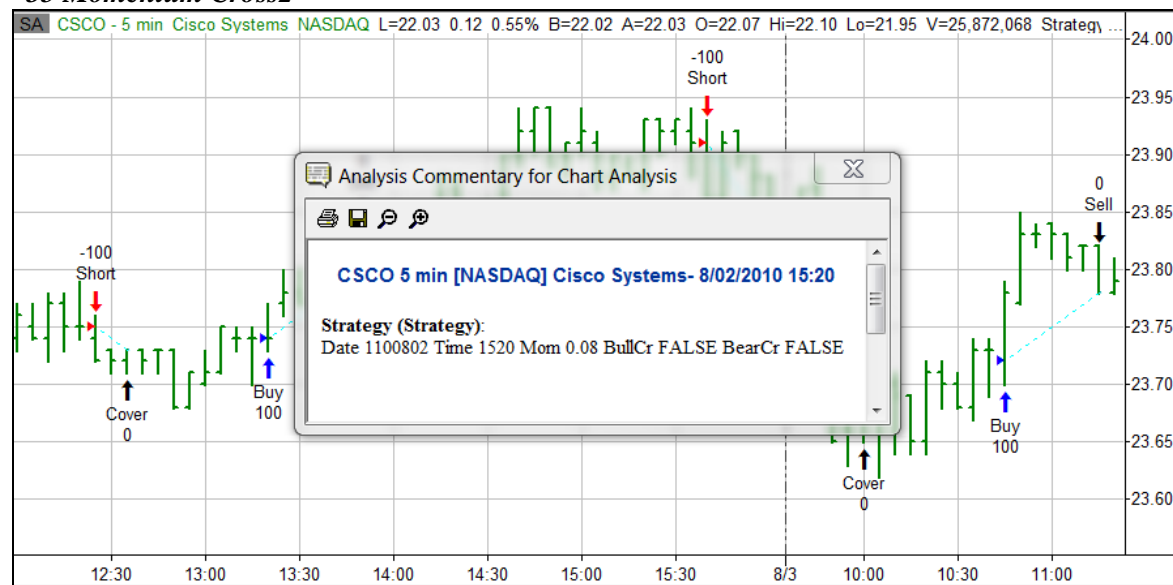
Commentary("Date", Spaces(2), Date:7:0, Spaces(2), "Time",
Spaces(2), Time:4:0, Spaces(2), "Mom", Spaces(2), Mom,
Spaces(2), "BullCr", Spaces(2), BullCr, Spaces(2), "BearCr",
Spaces(2), BearCr);
```

Verify the strategy and place it on a chart. You can see the information in the *Print Log*. Use the scroll bar on the right side of the *Print Log* to scroll to the top.



You can also view the information in the Analysis Commentary window by using the Analysis Commentary button in the toolbar  or by selecting *View* then *Analysis Commentary* from the main menu, and then clicking on one of the bars on the chart.

*35 Momentum Cross2



Congratulations on completing part one of the EasyLanguage Home Study Course! You are well on your way to mastering EasyLanguage and tapping into the power and potential of TradeStation.

The TradeStation Team

Appendix A

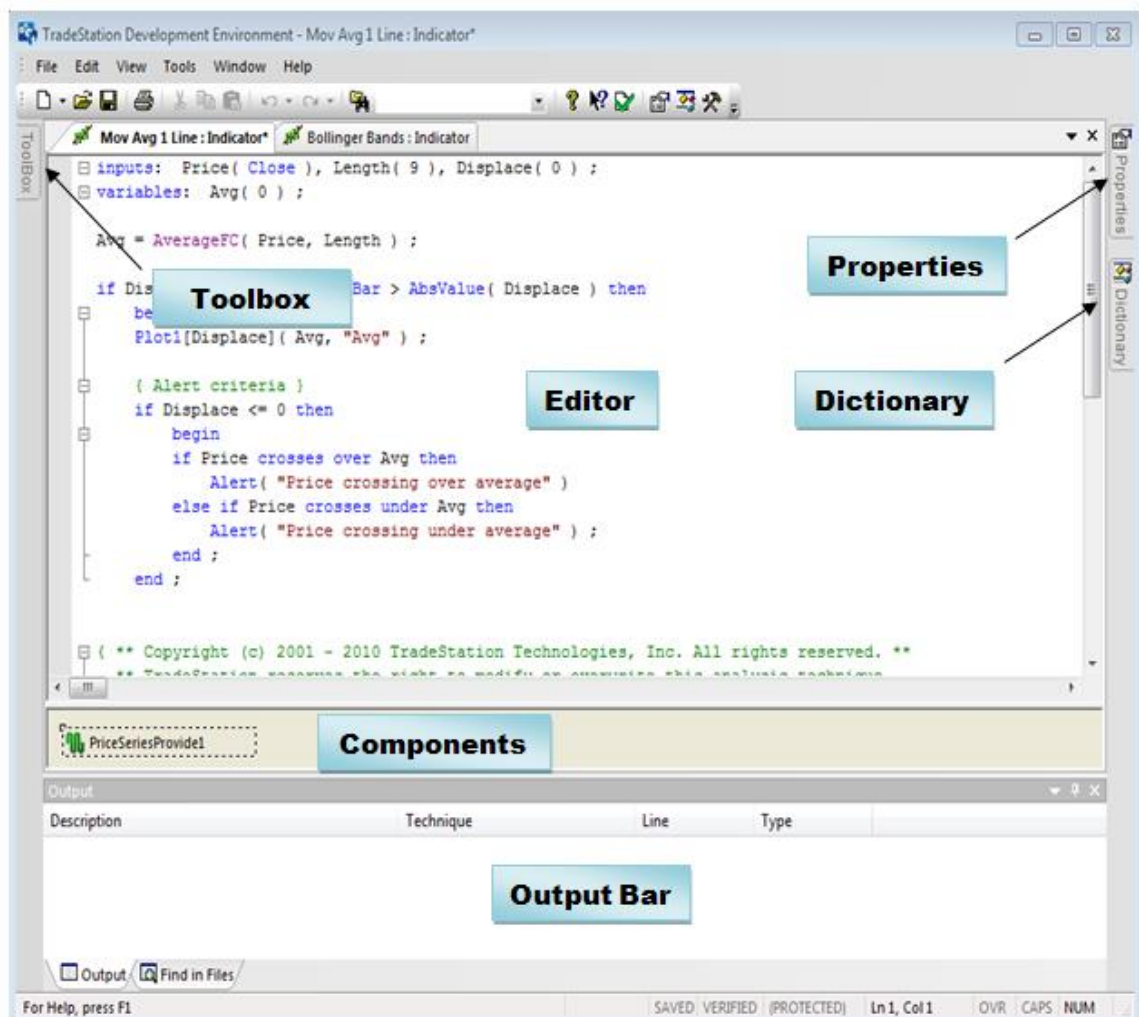
TradeStation Development Environment

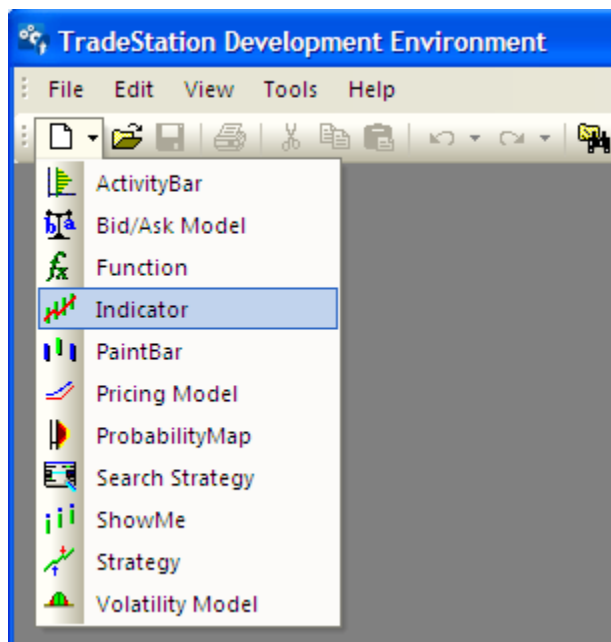
The TradeStation Development Environment offers a fully integrated set of tools for creating and editing your EasyLanguage documents, including analysis techniques, strategies, and functions.



The TradeStation Development Environment runs as a stand-alone application that opens independently from the TradeStation Platform. An icon appears in the TradeStation program group when clicking *Start – All Programs* from the Windows task bar. You may also click the EasyLanguage icon on the Tools section of the Shortcut Bar of the TradeStation Platform to launch the Development Environment.

The three main areas of the TradeStation Development Environment are the Editor, the Output Bar, and the Dictionary. The Editor is launched when a new EasyLanguage document is created or an existing one is opened. By default, the Output Bar is displayed at the bottom and the Dictionary is displayed as a retractable bar on the right.





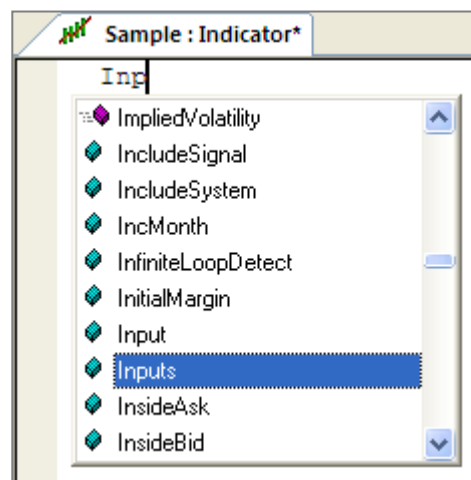
Creating EasyLanguage Documents

To create a new analysis technique or strategy in the TradeStation Development Environment, click the *File* – *New* menu sequence and choose the type of EasyLanguage document you would like to create from the menu. You may also click the *New* button, which is the first button displayed on the toolbar. The button has two clickable areas; clicking the white page will open the same type of EasyLanguage document last created, clicking the arrow will open a dropdown menu allowing you to choose the type of EasyLanguage document you want to create.

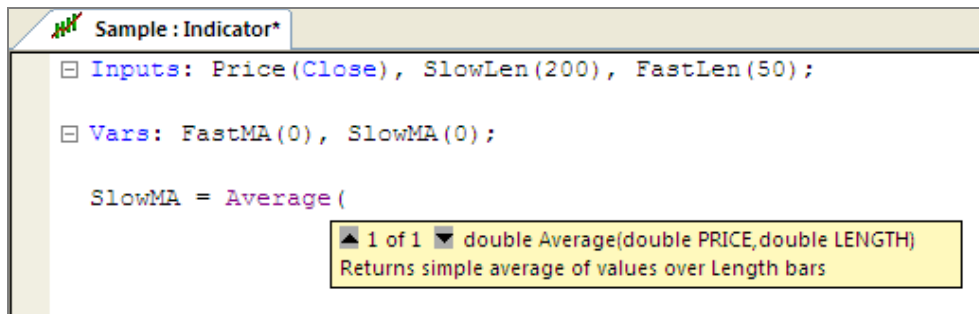
Autocomplete

Autocomplete monitors your typing and will prompt you with a pop-up window displaying a list of reserved words and functions that could be used to complete your entry. Based on the letters typed, the autocomplete window will show a list of all reserved words and functions matching your entry in alphabetical order. You may select from this window to help save time typing.

Once an input or variable has been declared in the document, the input or variable will also appear in the autocomplete list.



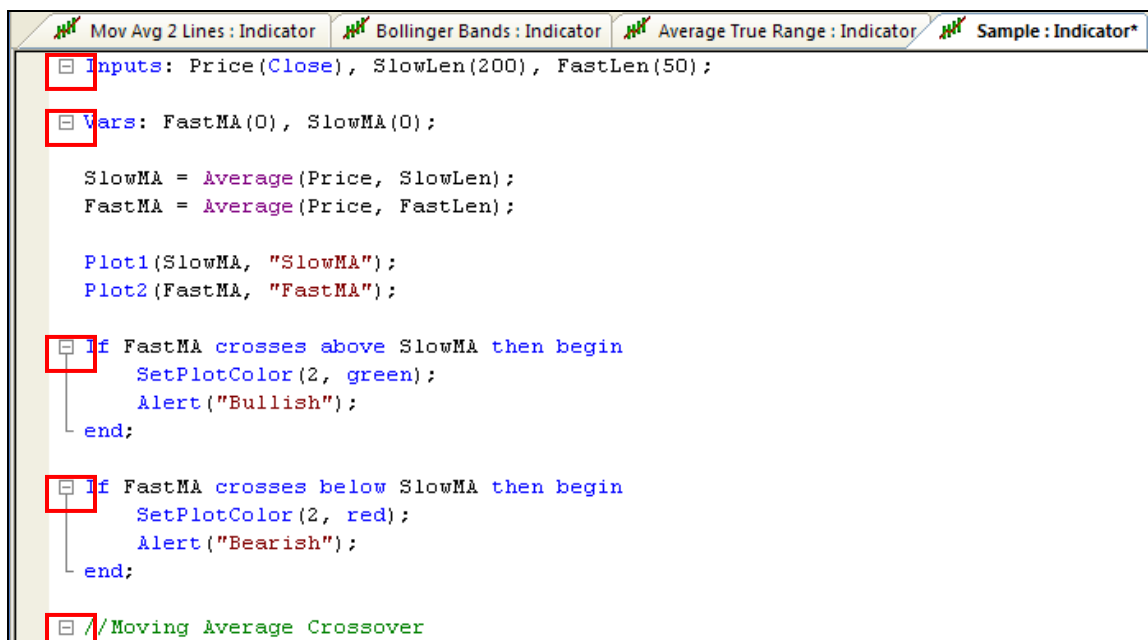
Autocomplete can also help you with the parameters for a reserved word or function. When you type a function into an EasyLanguage document and type the left parenthesis (or drag and drop it from the Dictionary), you will see a tool tip reminding you of the inputs and a brief description for the reserved word or function.

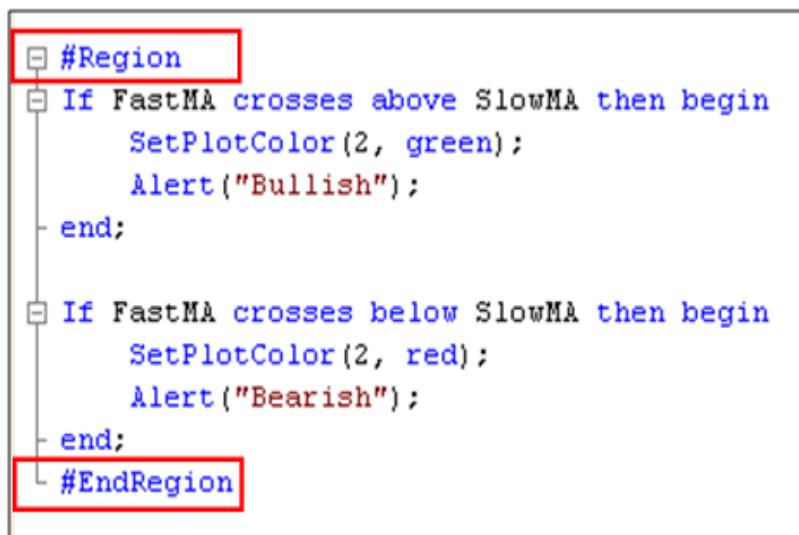


Note: The * next to the indicator name in the tab, tells you that the indicator is not saved.

Outlining

Outlining is a feature in the editor that allows you to collapse and expand sections of EasyLanguage. The outlining feature will automatically outline Inputs, Variables, Block If/Then statements, and comments. Clicking a minus sign on the left margin collapses a section; clicking a plus sign expands a section.



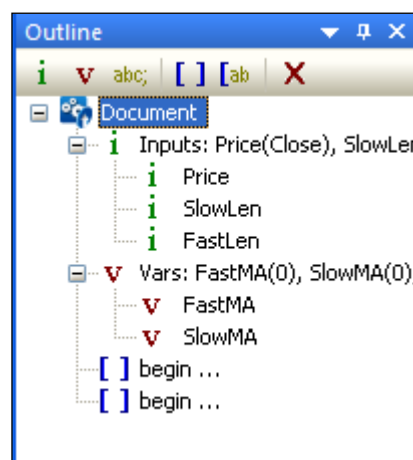


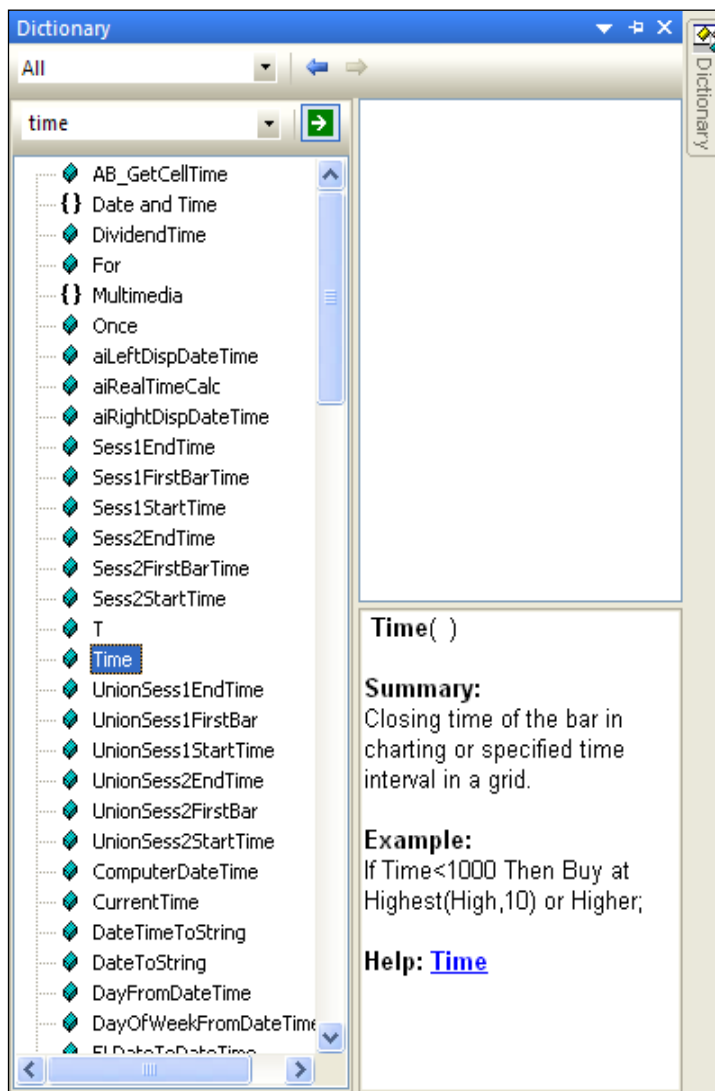
Custom Outlining

In the Development Environment, you may also customize outlining to define specific areas. Using the reserved word “#Region” at the beginning of the area and “#EndRegion” at the end, you may specify your own sections to outline. A new +/- box is created on the left margin next to “#Region” where you identified the beginning of the area to outline. This provides you with the ability to outline any area within a document and may help with organization and readability of the EasyLanguage document.

Outlining Toolbar

The Outlining toolbar is available to assist you in outlining. To display the Outlining toolbar in the TradeStation Development Environment, click the *View – Toolbars – Outline* menu sequence and the toolbar will appear on the left hand side. You can expand the outline by clicking the plus sign to see all of the Inputs, Variables and Block If...then statements that are in your document. Clicking any of the Inputs, Variables, or Block If...then statements in the toolbar will highlight the corresponding section in the Editor. The Outlining toolbar pictured on the right displays three Inputs, two Variables, and two Block If...then statements.



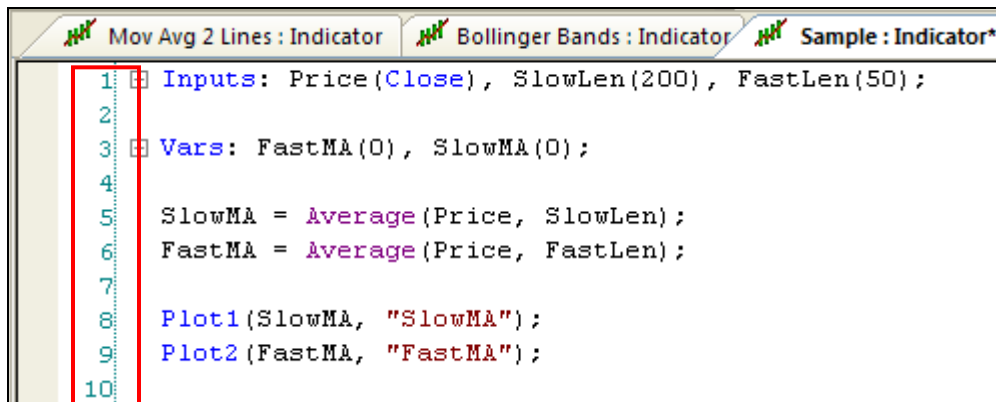


EasyLanguage Dictionary

On the right side of the Development Environment, a tab labeled *Dictionary* will give you access to the EasyLanguage Dictionary. The Dictionary can be a tremendous help while editing or creating strategies and analysis techniques. Clicking any of the categories listed in the Objects (left) pane will list members from that category in the Members (top right) pane. You may also search for reserved words or functions, by typing them in the *Search* field, and clicking the green arrow button or pressing Enter on the keyboard. Highlighting one of the words will display a short summary and example in the Description (bottom right) pane. Words and functions can be dragged and dropped directly into an EasyLanguage document. The picture on the left displays the results after searching for the word “time.” The EasyLanguage Dictionary simplifies the process of searching and finding the reserved words or functions needed.

Line Numbering

Line numbering is enabled by clicking *Tools - Options* from the menu bar. On the *General* tab, check *Line Numbers* and click *OK*. With this feature enabled, each line in the Editor will be numbered so you can easily identify specific lines where errors are identified.

A screenshot of the EasyLanguage Editor interface. The window has three tabs: 'Mov Avg 2 Lines : Indicator', 'Bollinger Bands : Indicator', and 'Sample : Indicator*'. The 'Sample : Indicator*' tab is active, displaying a script with 10 lines of code. A vertical red box highlights the line numbers on the left side of the editor, ranging from 1 to 10. The code is as follows:

```
1 Inputs: Price(Close), SlowLen(200), FastLen(50);  
2  
3 Vars: FastMA(0), SlowMA(0);  
4  
5 SlowMA = Average(Price, SlowLen);  
6 FastMA = Average(Price, FastLen);  
7  
8 Plot1(SlowMA, "SlowMA");  
9 Plot2(FastMA, "FastMA");  
10
```

Appendix B

Volume and Ticks

To a trader, volume refers to the number of shares or contracts traded, while ticks or tick count refers to the number of transactions. These words are also part of EasyLanguage, as are some variations on them such as UpTicks and DownTicks. Although their EasyLanguage meanings coincide with general trading terminology, they may be processed differently when applied to different types of charts and in RadarScreen.

These factors will affect how the words are processed:

Type of window:

- Chart Analysis
- RadarScreen

Type of symbol:

- Stock
- Electronic futures
- Forex

Chart interval:

- Intra-day
- Daily, weekly, monthly

Formatting:

- Time-, tick- or volume-based intra-day set to Trade Volume or Tick Count

For more detailed information please see the reference tables below. These tables are available for download at: www.TradeStation.com/support/training/downloads, under the file name, *Volume Reserved Words Reference Table*.

EasyLanguage Reserved Words Related to Ticks, Volume and Open Interest

Stock Symbols - Charting			
Chart Settings →	Intraday (Time-, Tick- or Volume-based Bars)		Daily (or greater)
Desired Value ↓	"Trade Volume" selected	"Tick Count" selected	
Volume	<i>Ticks</i>		<i>Volume or Ticks</i>
Tick Count		<i>Ticks</i>	
Volume on Up Ticks	<i>UpTicks</i>		
Volume on Down Ticks	<i>DownTicks</i>		
Up Tick Count		<i>UpTicks</i>	
Down Tick Count		<i>DownTicks</i>	

Stock Symbols - RadarScreen	
Desired Value ↓	All Intervals
Volume	<i>Volume or Ticks</i>

Note: Up Tick Count and Volume on Up Ticks include '0+' ticks. Down Tick Count and Volume on Down Ticks include '0-' ticks. The "Trade Volume" and "Tick Count" selections can be found in the Format Symbol dialog.

EasyLanguage Reserved Words Related to Ticks, Volume and Open Interest

Futures Symbols - Charting			
Chart Settings →	Intraday (Time-, Tick- or Volume-based Bars)		Daily (or greater)
Desired Value ↓	"Trade Volume" selected	"Tick Count" selected	
Volume	<i>Ticks</i>		<i>Volume</i>
Tick Count		<i>Ticks</i>	
Volume on Up Ticks	<i>UpTicks</i>		
Volume on Down Ticks	<i>DownTicks</i>		
Up Tick Count		<i>UpTicks</i>	
Down Tick Count		<i>DownTicks</i>	
Open Interest			<i>OpenInt</i>

Futures Symbols - RadarScreen		
Desired Value ↓	Intraday (Time- or Tick-based)	Daily (or greater)
Volume	<i>Volume or Ticks</i>	<i>Volume</i>
Open Interest		<i>OpenInt</i>

Note: Up Tick Count and Volume on Up Ticks include '0+' ticks. Down Tick Count and Volume on Down Ticks include '0-' ticks. The "Trade Volume" and "Tick Count" selections can be found in the Format Symbol dialog.

EasyLanguage Reserved Words Related to Ticks, Volume and Open Interest

Forex Symbols - Charting	
Chart Settings →	Intraday (Time- or Tick-based)
Desired Value ↓	"Tick Count" selected
Tick Count	<i>Ticks</i>
Up Tick Count	<i>UpTicks</i>
Down Tick Dount	<i>DownTicks</i>

Note: For these purposes, a bid change in a forex pair is considered a tick. Up Tick Count includes '0+' ticks. Down Tick Count includes '0-' ticks. The "Trade Volume" and "Tick Count" selections can be found in the Format Symbol dialog. None of these values are available for Forex symbols in RadarScreen.

Appendix C

Attending TradeStation Live In-Person Training Courses

The EasyLanguage Boot Camp is a two-day hands-on class designed to provide a solid working knowledge of EasyLanguage, with strong emphasis on practical information you can use right away. Our instructors will guide you step by step through exercises that cover creating strategies, indicators, and ShowMe™ and PaintBar™ studies, including studies designed specifically for use with RadarScreen®.

Note: This EasyLanguage Home Study Course book is based on the live EasyLanguage Boot Camp and utilizes many of the same examples.

The Mastering EasyLanguage for Strategies is a 2-day hands-on class specifically designed to follow the material covered in this course book or the EasyLanguage Boot Camp class. The purpose is to delve more deeply into using EasyLanguage for accomplishing a variety of tasks in creating, testing and automating strategies.

Topics include EasyLanguage specifically designed to take advantage of Intrabar Order Generation, multi-interval analysis, scaling out of positions, tying exits to entries, and much more. The teaching method will be familiar to those who have been through our other EasyLanguage classes: a task-oriented approach using practical examples.

We offer both these classes in computer-lab environments, providing you with a computer to use, with TradeStation already installed. This gives you a true hands-on opportunity to learn the essentials of EasyLanguage and how to use them in developing your own custom strategies and analysis techniques.

For information on class locations, availability and times, please call 800-808-3241 or visit <https://www.tradestation.com/support/training/courses> to view our calendar of live training events.

Appendix D

Importing the EasyLanguage Examples for this Course

All the EasyLanguage examples and exercises used in this course may be imported at:

<http://www.tradestation.com/ELHSC>

Course Challenge Videos

As mentioned earlier, all the Challenge answers and explanations are available as online videos with audio narration at:

<http://www.tradestation.com/ELHSC>

In addition, the answers are printed and explained in the next few pages and include sample charts for each Challenge.

Appendix E: Course Challenge Answers

Challenge 1: *03 High and Low

Learning objective: Using plot statements in indicators.

Write an indicator that plots a line connecting the highs of each bar and a second line connecting the lows of each bar.

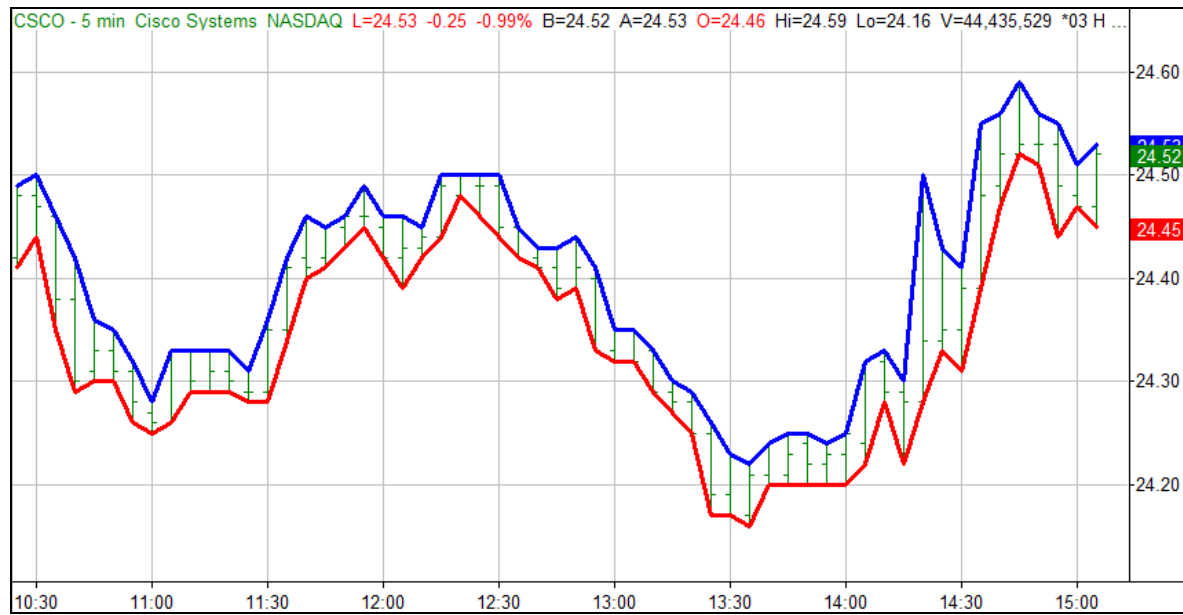
Set the properties to:

- Display the plots as lines, and

- Display the plots in the same sub-graph as the price data.

Hint: This indicator will require 2 plot statements. The reserved word in EasyLanguage for the high is `High`. The reserved word in EasyLanguage for the low is `Low`.

```
Plot1(High, "Hi");  
Plot2(Low, "Lo");
```



Challenge 2: *08 Bands

Learning objective: Using square brackets to reference previous values.

Write an indicator that plots two bands around the price data.

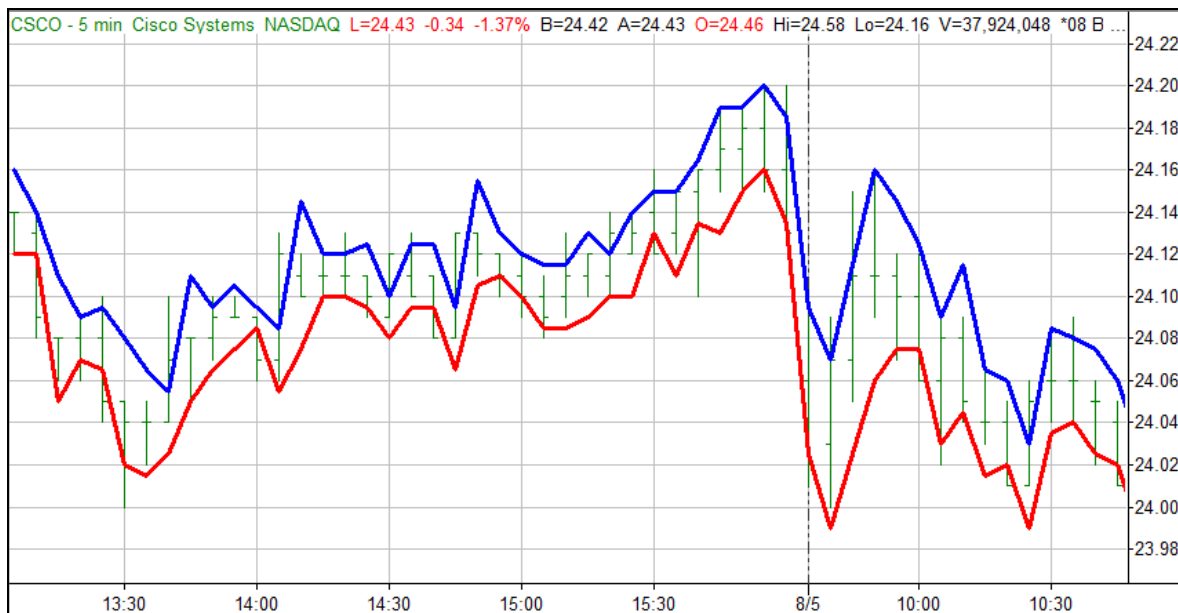
One band adds half the previous bar's range to the current bar's opening.

The other band subtracts half the previous bar's range from the current bar's opening.

Remember: The range of a bar is high minus low. How would you express the range of the previous bar?

This indicator should overlay the price bars; therefore the *Scale On* setting on the *Scaling* tab in the *Indicator Properties* should be set to *Same Axis as Underlying Data*.

```
Plot1(Open + (High[1] - Low[1]) *.5, "HighBand");  
Plot2(Open - (High[1] - Low[1]) *.5, "LowBand");
```



Challenge 3: *13 Envelope

Learning objective: Calling a function into an EasyLanguage document; using user-declared numeric variables.

Write an indicator that plots a 20-bar moving average of the highs and a 20-bar moving average of the lows.

Select names and declare variables for the two moving average values. This will require variable assignment statements, too.

This indicator should plot in the same sub-graph as the price data.

Hint: The `Average` function may be dragged in from the Dictionary.

```
Vars: UpperLine(0), LowerLine(0);

UpperLine = Average(High, 20);
LowerLine = Average(Low, 20);

Plot1(UpperLine, "UpperLine");
Plot2(LowerLine, "LowerLine");
```



Challenge 4: *19 Weak Close

Learning objective: Writing a ShowMe study; using block **If...then...else** and **NoPlot** statements.

Write a ShowMe that marks the high of those bars that have a close that is in the bottom third of their range. Include a **NoPlot statement to remove a **Plot** if a true condition becomes false before bar close.**

Include instructions to be alerted when this condition occurs.

The range of the bar is $\text{High} - \text{Low}$, but there is also a function called **Range** that makes this calculation for you.

```
Input: RangePortion(3);

If Close < Low + (Range / RangePortion) then begin
    Plot1(High, "Weak Close");
    Alert;
end
else
    NoPlot(1);
```



Challenge 5: *22 Parts Of Day

Learning objective: Writing a PaintBar study using Time and logical operators.

Write a PaintBar that paints the lower half of those bars that are in the first and last hours of the day.

Declare Inputs for the times that delineate the first and last hours of the day.

```
Inputs: FirstPartEnd(1030), LastPartStart(1501);
```

```
If Time <= FirstPartEnd or Time >= LastPartStart then  
  PlotPB(Low, (High + Low) * .5, "PartsOfDay");
```



Challenge 6: *26 Key Reversal

Learning objective: Using user-declared true/false variables; describing bar patterns; limit order syntax.

Write a strategy that uses key reversals up and key reversals down to identify entry points.

Declare and assign variables for key reversals up and down.

Have the strategy enter a limit order to buy on the bar following a key reversal up, at a limit price better than the current bar's close.

Have the strategy enter a limit order to sell short on the bar following a key reversal down, at a limit price better than the current bar's close.

Declare an input for the number of points above or below the reversal bar's close to set the limit order prices; have the input default to 5 points.

```
Input: LimitPoints(.05);

Vars: RevUp(false), RevDown(false);

RevUp = Low < Low[1] and Close > Close[1];
RevDown = High > High[1] and Close < Close[1];

If RevUp then
    Buy next bar at Close of this bar - LimitPoints limit;

If RevDown then
    SellShort next bar at Close of this bar + LimitPoints
    limit;
```

