Wiley Trading

BUILDING WINNING TRADING SYSTEMS with TradeStation™

George Pruitt and John R. Hill
Futures Truth

CRACKED TRADING SOFTWARE

70+ DVD's FOR SALE & EXCHANGE

www.traders-software.com

www.forex-warez.com

www.trading-software-collection.com

www.tradestation-download-free.com

Contacts

andreybbrv@gmail.com
andreybbrv@yandex.ru
Skype: andreybbrv

Building Winning Trading Systems with TradeStation™

Founded in 1807, John Wiley & Sons is the oldest independent publishing company in the United States. With offices in North America, Europe, Australia and Asia, Wiley is globally committed to developing and marketing print and electronic products and services for our customers' professional and personal knowledge and understanding.

The Wiley Trading series features books by traders who have survived the market's ever changing temperament and have prospered—some by reinventing systems, others by getting back to the basics. Whether a novice trader, professional, or somewhere in between, these books will provide the advice and strategies needed to prosper today and well into the future.

For a list of available titles, please visit our Website at www.WileyFinance.com.

Building Winning Trading Systems with TradeStation™

George Pruitt
John R. Hill
FUTURES TRUTH

John Wiley & Sons, Inc.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey
Published simultaneously in Canada

Library of Congress Cataloging-in-Publication Data:
Pruitt, George, 1967–
Building winning trading systems with TradeStation™ / George Pruitt, John R. Hill.
p. cm. — (the Wiley trading series)
Includes index.
ISBN 0-471-21569-4 (cloth: alk. paper)
1. Investments—Data processing. 2. Stocks—Data processing. I. Hill, John R., 1926– II. Title. III. Series.
HG4515.95.P78 2002
332.64'2'02855369—dc21

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

To my loving wife and family.
J. H.

I would like to dedicate this book to Mary, Cliff, Butch, and Marilyn for their eternal courage and support. I would also like to thank my loving wife, Leslie and my patient and understanding children, Brandon and Emily.
G. P.

Contents

Acknowledgments

We would like to thank TradeStation Securities, OptionVue software, and Jan Arps and Len Yates for their contributions.

Introduction

Before System Writer (the grandfather of today's current TradeStation), system developers and traders did not have a commercialized software platform to develop their trading ideas. We are not talking about charting packages; we are talking about programs that could understand a trading strategy based on technical analysis. Sure, there were other sophisticated programs in the 1980s that could be used, but they required a thorough understanding of programming, additional software, and extreme patience with the software developer. Most of these programs were not open platform; you had to program within their limited scripting language and could not share your programs with others. The programs that were potentially open platform usually required a separate editor and compiler and a very sophisticated user. In fact, we developed and still use a package that is based on a FORTRAN compiler. The System Writer/TradeStation programs provided one sleek package that gave the ability to test, optimize, and trade to the trading masses. TradeStation has been accepted worldwide as the premier market analysis platform and the standard against which other trading/testing platforms are measured. The latest version of TradeStation has evolved into more than just software; it is a fully self-contained trading platform that incorporates direct access brokerage for futures and equities traders. Along with the birth of TradeStation 6.0, Omega Research has been reborn and renamed to TradeStation Securities.

We are not here to proclaim TradeStation's Securities products to be the best in the business, but we are here to educate their users on how to use what is accepted as the industry standard. Of course Omega Research has had their problems; all software developers have had their problems. Recently, it was heard that the inventor of the CTRL + ALT + DEL key sequence at IBM

stated that he may have created it, but Microsoft made it famous. What sets TradeStation apart from the other software packages is its powerful scripting language EasyLanguage. EasyLanguage is more powerful than easy; it really isn't a scripting language, but more of a full-blown programming language. It can be compared to BASIC, FORTRAN, Pascal, or C. In fact, it is based off of a Pascal compiler and has been around since 1987. Sam Tennis, the father of EasyLanguage, wanted to provide traders with a simple and logical language that required little programming knowledge. In our opinion, he was successful in this endeavor.

Technical analysis of stocks and commodities is complex and you would think that a programming language that deals with such a lofty subject would be as complex. Fortunately, EasyLanguage comes with a complete library of the industry's most widely used analysis techniques. Third-party developers have further extended this library. Traders do not need to recreate the wheel, nor do they need the programming knowledge to put these techniques into action. Traders can even customize their own ideas or existing ones and add them to the library.

This book is designed for all TradeStation and EasyLanguage users. However, this book does expect users to be somewhat familiar with TradeStation and its functions. Beginners can obtain a good foundation on programming technique, program control structures, data structures, and familiarization on the use of the EasyLanguage built-in functions. All users will benefit from the chapters that discuss proper trading system development. We explore all areas of analysis techniques from Indicators to Paint Bars with a special emphasis on trading strategies.

Since a large portion of this book involves actual computer code, a companion CD-ROM, with all of the computer programs and data, is provided for the reader. The analysis techniques are provided in TradeStation 2000i and TradeStation 6.0 formats. Previous version users can still utilize the analysis techniques by simply typing them into their version of PowerEditor. Chapter 5 enlists the help of Microsoft Excel spreadsheet software to create three-dimensional contour charts. Excel isn't necessary, but some type of spreadsheet software is needed to transform the data into a chart.

This book was designed with the trader in mind. Most of the trading techniques were designed and tested with indices, futures, and commodities (we focused on these markets due to their long histories with trading systems). Strictly equity traders will still get a good programming and good system design education. Index (mini or full-size) and futures traders will be privy to five highly successful trading approaches. These approaches are good launching pads for further and much more detailed research.

In our opinion, the best way to make full use of this book is for the reader to work his way through the book starting with Chapter 1. At the end of chapters that have detailed *analysis techniques* (or *computer programs*—the two terms

are interchangeable), we would suggest loading from the CD-ROM, verifying, and running the programs in TradeStation. The concepts of each chapter should be mastered before moving on to the next chapter.

We hope you (the reader) enjoy this book and that it opens your eyes to the power of TradeStation and your own creativity. Good luck and good trading.

1

Fundamentals

WHAT IS EASYLANGUAGE?

When you code (slang for writing your ideas into a programming language) an analysis technique, you are directing the computer to follow your instructions to the tee. A computer program is nothing but a list of instructions. A computer is obedient and speedy, but it is only as smart as its programmer. In addition, the computer requires that its instructions to be in an exact format. A programmer must follow certain syntax rules.

EasyLanguage is the medium used by traders to convert a trading idea into a form that a computer can understand. Fortunately for nonprogrammers, EasyLanguage is a high level language; it looks like the written English language. It is a compiled language; programs are converted to computer code when the programmer deems necessary. The compiler then checks for syntactical correctness and translates your source code into a program that the computer can understand. If there is a problem, the compiler alerts the programmer and sometimes offers advice on how to fix it. This is different than a translated language, which evaluates every line as it is typed.

All computer languages, including EasyLanguage, have several things in common. They all have:

- Reserved Words. Words that the computer language has set aside for a specific purpose. You can only use these words for their predefined purposes. Using these words for any other purpose may cause severe problems. (See the list of reserved words in Appendix C.)

- Remarks. Words or statements that are completely ignored by the compiler. Remarks are placed in code to help the programmer, or other people who may reuse the code, understand what the program is designed to do. EasyLanguage also utilizes *skip words*. These words are included in a statement to make the programming easier to read. For example, *Buy on next bar at myPrice stop* is the same as *Buy next bar myPrice stop*. The words *on* and *at* are completely ignored. (See the list of skip words in Appendix C.)
- Variables. User-defined words or letters that are used to store information.
- Data Types. Different types of storage; variables are defined by their data types. EasyLanguage has three basic data types: Numeric, Boolean, and String. A variable that is assigned a numeric value, or stored as a number, would be of the Numeric type. A variable that stores a true or false value would be of the Boolean type. Finally, a variable that stores a list of characters would be of the String type.

Variables and Data Types

A programmer must understand how to use variables and their associated data types before they can program anything productive. Let's take a look at a snippet of code.

```
mySum = 4 + 5 + 6;
myAvg = MySum/3;
```

The variables in this code are *mySum* and *myAvg* and they are of the Numeric data type; they are storage places for numbers. EasyLanguage is liberal concerning variable names, but there are a few requirements. A variable name cannot

- start with a number or a period (.)
- be a number
- be more than 20 alphanumeric characters long
- include punctuation other than the period (.) or underscore (_)

Correct	Incorrect
myAvg	1MyAvg
mySum	.sum
sum	val+11
val1	the//sum
the.sum	my?sum
my_val	1234

Variable naming is up to the style of the individual programmer. EasyLanguage is not case sensitive (you can use upper or lowercase letters in the variable names). (Note: This is our preference—may not be everybody's.) Lowercase letters are preferred for names that only contain one syllable. For variable names that have more than one syllable, we begin the name with a lowercase letter and then capitalize the beginning of each subsequent syllable.

```
sum, avg, total, totalSum, myAvg, avgValue, totalUpSum, totDnAvg
```

Still referring to the previous snippet of code, mySum is assigned the value of 15 and myAvg is assigned the value of 15/3 or 5. If a variable name is created, it must be declared ahead of time. The declaration statement defines the initial value and data type of the variable. The compiler needs to know how much space to reserve in memory for all variables. The following code is a complete EasyLanguage program. (Note: Most of the code that you will see in this book will be particular to EasyLanguage and will probably not work in any other language.)

```
Vars: mySum(0),myAvg(0);
mySum = High + Low + Close ;
myAvg = mySum/3 ;
```

The *Vars:* (or Variables:) statement tells the computer what variables are being declared and initialized. We declare the variables by simply listing them in the Vars statement and initialize them by placing an initial value in parentheses following the variable name. In this case, mySum and myAvg are to be equal to zero. EasyLanguage is smart enough to realize that these variables should be of the Numeric data type, since we initialized them with numbers. Variable names should be self-descriptive and long enough to be meaningful. Which of the following is more self-explanatory?

```
mySum = High+Low+Close;        or      k = High + Low + Close;
myAvg = mySum/3;                       j = k/3;
BuyPt = Close + myAvg;                 l = Close+j;
```

Variables of Boolean and String types are declared in a similar fashion.

```
Vars: myCondition(false),myString("abcdefgh");
```

The variable *myCondition* was initialized to false. The word *false* is a reserved word that has the value of zero. This word cannot be used for any other purpose. The variable *myString* was initialized to abcdefgh. Sometimes you will need to use a variable for temporary purposes and it is difficult to declare and initialize all of your variables ahead of time. In the case of a temporary variable (one that holds a value for a short period of time), EasyLanguage has declared and initialized several variables for your use; value0 through value99 have been predefined and initialized to zero and are ready for usage in your programs. The following is a complete EasyLanguage program:

```
value1 = High + Low + Close;
value2 = (High + Low)/2.0;
```

Notice there isn't a Vars statement. Since value1 and value2 are predefined, the statement isn't needed. You have probably noticed the semicolon (;) at the end of each line of code. The semicolon tells the compiler that we are done with this particular instruction. In programming jargon, instructions are known as *statements*. Statements are made up of expressions, which are made up of constants, variables, operators, functions, and parentheses. Some languages need a termination symbol and others do not. EasyLanguage needs the statement termination symbol. Remember to put a semicolon at the end of each line to prevent a syntax error.

Inputs are similar to variables. They follow the same naming protocol and are declared and initialized. However, an input remains constant throughout an analysis technique. An input cannot start a statement (a line of instruction) and cannot be modified within the body of the code. One of the main reasons for using inputs is that you can change input values of applied analysis techniques without having to edit the actual EasyLanguage code. Inputs would be perfect for a moving average indicator. When you plot this indicator on a chart, you simply type in the length of the moving average into the input box of the dialog box. You don't want to have to go back to the moving average source code and change it and then verify it. Also, when used in trading strategies, inputs allow you to optimize your strategies. This is discussed in Chapter 5.

Notice how inputs and variables are declared in similar style.

```
Inputs: length1(10),length2(20),flag(false);

Vars:   myLength1(10),myAvgVal(30);
```

However, notice how they are used differently in coding.

Variables

```
myLength1 = myAvgVal + myLength1;  {Correct}
```

Inputs

```
length1 = myAvgVal + length1;      {Incorrect}
myLength1 = length1*2;             {Correct}
```

Variables can start a statement and can be assigned another value. Since inputs are constants and cannot be assigned new values, they cannot start a statement.

In a strongly typed language, such as C, Pascal, or C++, if you assign a real value such as 3.1456 to an integer typed variable, the decimal portion is truncated and you end up with the number 3. As we all know, precision is important when it comes to trading, so EasyLanguage includes only one Numeric type. All numbers are stored with a whole and a fractional part. In the old days when CPUs were slow, noninteger arithmetic took too much time and it was advised to use integer variables whenever possible.

OPERATORS AND EXPRESSIONS

Previously, we discussed statements and how they are made up of expressions. To review, an expression consists of a combination of identifiers, functions, variables, and values, which result in a specific value. Operators are a form of built-in functions and come in two forms: unary and binary. A binary operator requires two operands, whereas a unary operator requires only one. Most of your dealings with operators in EasyLanguage will be of the binary variety. Some of the more popular ones are: + – / * < = > >= <= <> AND OR. These binary operators can be further classified into two more categories: arithmetic and logical.

Expressions come in three forms: arithmetic, logical, and string. The type of operator used determines the type of expression. An arithmetic expression includes + – / *, whereas a logical or Boolean expression includes < = > >= <= <> AND OR.

Arithmetic Expressions	**Logical Expressions**
myValue = myValue + 1;	myCondition1 = sum > total;
myValue = sum – total;	myCondition1 = sum <> total;
myResult = sum*total+20;	cond1 = cond1 AND cond2

Arithmetic expressions always result in a number, and logical expressions always result in true or false. True is equivalent to 1, and False is equivalent to 0. String expressions deal with a string of characters. You can assign string values to string variables and compare them.

```
myName1 = "George Pruitt";
myName2 = "John Hill";
cond1 = (myName1 <> myName2);
myName3 = myName1 + " " + myName2;
```

Concatenation occurs when two or more strings are added together. Basically, you create one new string from the two that are being added together.

PRECEDENCE OF OPERATORS

It is important to understand the concept of precedence of operators. When more than one operator is in an expression, the operator with the higher precedence is evaluated first and so on. This order of evaluation can be modified with the use of parentheses. EasyLanguage's order of precedence is as follows:

1. Parentheses
2. Multiplication or division
3. Addition or subtraction
4. <, >, =, <=, >=, <>
5. AND
6. OR

Here are some expressions and their results:

1.	20 – 15/5	equals 17 not 1
	20 – 3	division first, then subtraction
2.	10 + 8/2	equals 14 not 9
	10 + 4	division first then addition
3.	5 * 4/2	equals 10
	20/2	division and multiplication are equal
4.	(20 – 15)/5	does equal 1
	5/5	parentheses overrides order
5.	(10 + 8)/2	equals 9
	18/2	parentheses overrides order
6.	6 + 2 > 3	true
	8 > 3	
7.	2 > 1 + 10	false
	2 < 11	

8.	2 + 2/2 * 6	equals 8 not 18
	2 + 1 * 6	division first
	2 + 6	then multiplication
	8	then addition

These examples have all the elements of an arithmetic expression—numeric values and operators—but they are not complete EasyLanguage statements. An expression must be part of either an assignment statement (myValue = mySum + myTot) or a logical statement (cond1 = cond2 OR cond3).

The overall purpose of EasyLanguage is to translate an idea and perform an analysis on a price data series over a specific time period. A price chart consists of bars built from historical price data. Each individual bar is a graphical representation of the range of prices over a certain period of time. A five-minute bar would have the Opening, High, Low, and Closing prices of an instrument over a five-minute time frame. A daily bar would graph the range of prices over a daily interval. Bar charts are most often graphed in an Open, High, Low, and Close format. Sometimes the opening price is left off. A candlestick chart represents the same data, but in a different format. It provides an easier way to see the relationship between the opening and closing prices of a bar chart. Other bar data such as the date and time of the bar's close, volume, and open interest is also available for each bar. Since EasyLanguage works hand-in-hand with the charts that are created by TradeStation, there are many built-in reserved words to interface with the data. These reserved words were derived from commonly used verbiage in the trading industry. You can interface with the data by using the following reserved words. (Note: Each word has an abbreviation and can be used as a substitute.)

Reserved Word	**Abbreviation**	**Description**
Date	D	Date of the close of the bar.
Time	T	Time as of the close of the bar.
Open	O	Open price of the bar.
High	H	High price of the bar.
Low	L	Low price of the bar.
Close	C	Close price of the bar.
Volume	V	Number of contracts/shares traded.
OpenInt	OI	Number of outstanding contracts (futures only).

If you wanted to determine that the closing price of a particular instrument was greater than its opening price you would simply type: Close > Open, or, C > O.

The beauty of EasyLanguage is its ability to have all of the data of an instrument at your fingertips. The reserved words that we use to access the different prices of the current bar are also used to access historical data. You do this by adding an index to the reserved word. The closing price of yesterday would be: Close[1]. The closing price two days ago would be: Close[2] and so on. The number inside the bracket determines the number of bars to look back. The larger the number, the further you go back in history. If you wanted to compare today's closing price with the closing price ten days prior you would type: Close > Close[10].

Before we move on, we should discuss how TradeStation stores dates and times. January 1, 2001 is stored as 1010101 instead of 20010101 or 010101. When the millennium changed, instead of incorporating the century into the date, TradeStation simply added a single digit to the year. The day after 991231 was 1000101 according to TradeStation. Time is stored as military time. For example, one o'clock in the afternoon is 1300 and one o'clock in the morning is 100.

After that brief introduction to the world of programming, let's go ahead and set up and program a complete trading strategy. The PowerEditor is where all of the magic takes place. This is your interface between your ideas and their applications. All of your coding takes place here, and a thorough understanding of this editor will reduce headaches and increase productivity. In TradeStation 4.0 and 2000i, the PowerEditor is basically a stand-alone application; it is an independent program and can be run with or without TradeStation. In the newer TradeStation 6.0, the PowerEditor is more of a component program; it runs within the confines of TradeStation.

TRADESTATION 2000i VERSUS TRADESTATION 6.0

As of the writing of this book, there are two versions of TradeStation currently being used: 2000i and 6.0. Based on input from users, we figure that half is using one version and the other half is using the other. For this reason, the remainder of this chapter will be broken into two main sections. (Beyond this chapter, however, we will include the EasyLanguage code for 2000i in Appendix B. We will concentrate on 6.0 in the remaining chapters as it is the latest version and seems to be the way of the future.) Version 6.0 and 2000i are different enough to merit treating each one separately and we will refer to the two versions as 6.0 and 2000i. TradeStation 6.0 is Omega Research's all-inclusive trading tool. Everything that you need to design, test, monitor, and execute an analysis technique is in one slick and complete package. Omega Research is now a brokerage/software company and equities and futures trades can be executed through direct access with TradeStation.

TradeStation 2000i

PowerEditor

Once this program is up and running, go under the File menu and select New. A dialog box titled *New* will open. If the General tab is not selected, go ahead and select it. Your screen should look like this:

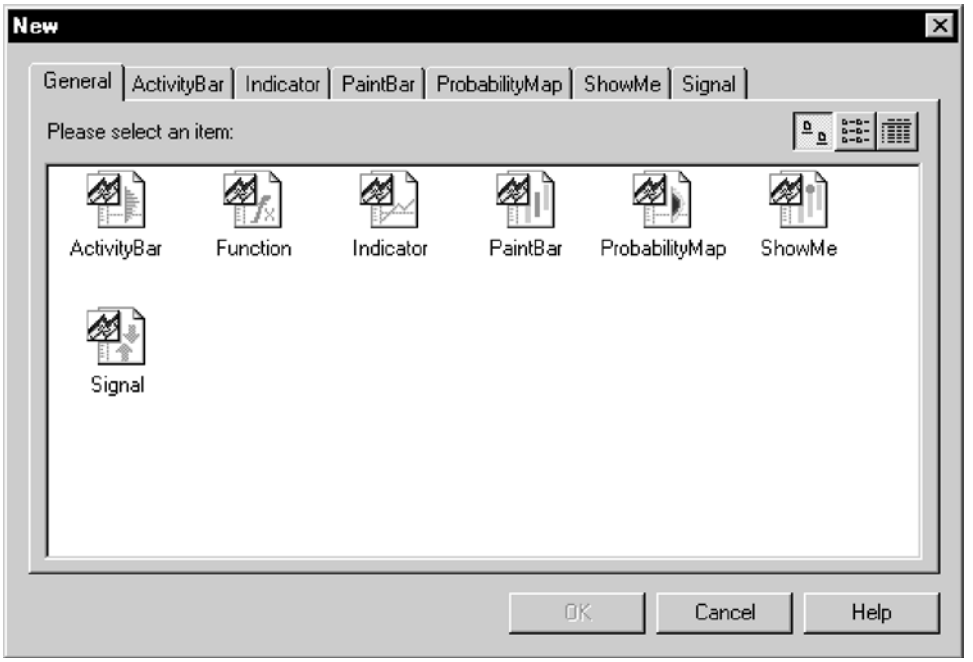

Figure 1.1 New Dialog—TradeStation 2000i

Once your screen looks like Figure 1.1 select the icon with the title Signal and click OK. We will ignore the other tabs in this dialog box at the moment. Another dialog box titled *Create a New Signal* will open and ask for a name and notes about the signal that you are creating. In the Name field go ahead and type "MySignal-1" and in the Notes field type "Donchian Break Out" and then click OK. A window titled *MySignal-1* should open. This is your clean sheet of paper on which to type your wonderful ideas. Before we do some coding, let's briefly take a look at some of the menus. Table 1.1 details the selections that are available under the File menu.

Table 1.2 details the selections under the Edit menu.

Table 1.3 details the selections under the View menu.

Table 1.4 details the selections under the Tool menu.

Table 1.1
File Menu

Menu Item	Action
New	Creates a new blank analysis technique.
Open	Opens an existing analysis technique.
Close	Closes current window.
Save	Saves the current analysis technique.
SaveAs	Allows renaming of the current analysis technique.
SaveAs Template	Saves the current analysis technique as a template for future use.
Save All	Saves all currently open analysis techniques.
Import and Export	Imports existing analysis techniques from other sources or exports analysis techniques to other sources.
Protect	Protects the analysis technique with a password.
Verify	Verifies the analysis technique. Checks for syntax errors in code.
Verify All	Verifies all functions and analysis techniques.
Properties	Shows the name and notes of the analysis technique. Allows the user to change these.
Page Setup	Allows changing of the page setup.
Print	Prints the code.
Print Preview	Shows what will be printed.
Exit	Exits out of PowerEditor.

Table 1.2
Edit Menu

Menu Item	Action
Undo	Undoes the last action
Redo	Redoes the last action.
Cut	Cuts the selected text.
Copy	Copies the selected text.
Paste	Pastes the selected text.
Clear	Clears the selected text.
Select All	Selects all text in the analysis technique.
Find	Finds a specific string of text.
Find Next	Finds the next occurrence of the specified string of text.
Find in Files	Powerful multifile search tool.
Replace	Replaces a string of text with another string of text.

Table 1.3
View Menu

Menu Item	Action
Toolbars	Customizes the tool bars.
Status Bar	Hides/shows the Status bar.
Output Bar	Hides/shows the Output bar. This little window will become important when we start debugging.
Bookmarks	Marks a particular section of text for reference purposes
Font	Sets the PowerEditor's font.
Options	Displays options for the PowerEditor.

Table 1.4
Tool Menu

Menu Item	Action
EasyLanguage Dictionary	Inserts EasyLanguage components into an analysis technique.
Errors Window Options	Changes the look of the Errors window.
Find In Files Window Options	Changes the look of the Find In Files window.
Debug Window	Changes the attributes of the Debug window.

A Simple Program

Now that we are familiar with the menus and their functions, let's go ahead and code a simple program. We don't need to know everything about the selections in the menus to start coding. Jump in headfirst and type the following text exactly as you see it here:

```
Inputs: longLength(40), shortLength(40);

Buy tomorrow at Highest(High,longLength) stop;
Sell tomorrow at Lowest(Low,shortLength) stop;
```

(Note: for those of you who may be moving to TradeStation 6.0 you must type Sell Short to initiate a short position.)

After you have typed this, go under the File menu and select Verify or hit the F3 key. Many of the commands in the menus have keyboard equivalents or shortcuts. You can determine the shortcuts by selecting the menu and then the menu item. (The keyboard shortcut is listed to the far right of the menu item.) If you look at the Verify menu item on the File menu, you will see F3. You should get a small dialog box that first says *Verifying* and then *Excellent*. If you

get an error, simply check your code for any typos and Verify again. Congratulations, you have just written an analysis technique in the form of a signal! Now let's break each line of code down so that we can fully understand what is going on.

```
Inputs: longLength(40), shortLength(40);
```

By typing this line you have created two constant variables of the numeric data type. These two variables, longLength and shortLength, have been initiated with the value of 40. These variables cannot be changed anywhere in the body of the analysis technique. They can only be changed from the user interface of this signal or in the program heading. This will be discussed later in this chapter.

```
Buy tomorrow at Highest(High,longLength) stop;
```

This line instructs the computer to place a buy stop tomorrow at the highest high of the last 40 days. Highest is a function call. Functions are subprograms that are designed for a specific purpose and return a specific value. To communicate with a function, you must give it the information it needs. An automatic teller machine is like a function; you must give it your PIN before it will give you any money. In the case of the Highest function, it needs two bits of information: what to look at and how far to look back. We are instructing this function to look at the highs of the last 40 bars and return the highest of those highs. For now, just accept that High means High prices and Low means Low prices. (This is yet another subject that will be touched upon later.) When an order is instructed through EasyLanguage, you must tell the computer the type of order. In this case, we are using a stop order. Orders that are accepted by EasyLanguage are:

- *Stop*. Requires a price and is placed above the market to buy and below the market to sell.
- *Limit*. Requires a price and is placed below the market to buy and above the market to buy.
- *Market*. Buys/sells at the current market price.

So, if the market trades at a level that is equal to or greater than the highest high of the past forty days, the signal would enter long at the stop price.

```
Sell tomorrow at Lowest(Low,shortLength) stop;
```

The instructions for entering a short position are simply the opposite for entering a long position.

TradeStation StrategyBuilder

Now let's create a trading strategy with our simple Donchian Break Out signal. The StrategyBuilder is a program that asks all of the pertinent information concerning a trading strategy. It helps to organize all of the different trading ideas and makes sure all of the parameters are set before the signal is evaluated. Under the Go menu, select TradeStation StrategyBuilder. A dialog box titled *TradeStation StrategyBuilder* should open and look similar to Figure 1.2.

Click on the New button. Another dialog box opens and asks for a name and notes for this strategy. In the Name field type, "MyStrategy-1", and in the Notes field type, "A simple Donchian Break Out". After typing the information into these fields click the Next button. The next dialog box asks for the signal to be used in this strategy. Click on the Add button. The next dialog box should look similar to Figure 1.3.

Scroll up/down and find MySignal-1. You will notice that the boxes under the Long Entry and Short Entry column headings are checked, but the boxes underneath the Long Exit and Short Exit are not. This tells us that the system only enters the market; it is never flat—the system is either long or short. Long positions are liquidated when a short position is initiated and vice versa. Our simple Donchian Break Out is a pure stop and reverse system. Click on MySignal-1 and click on the OK button. Click the next button and a dialog box like the one in Figure 1.4 will open.

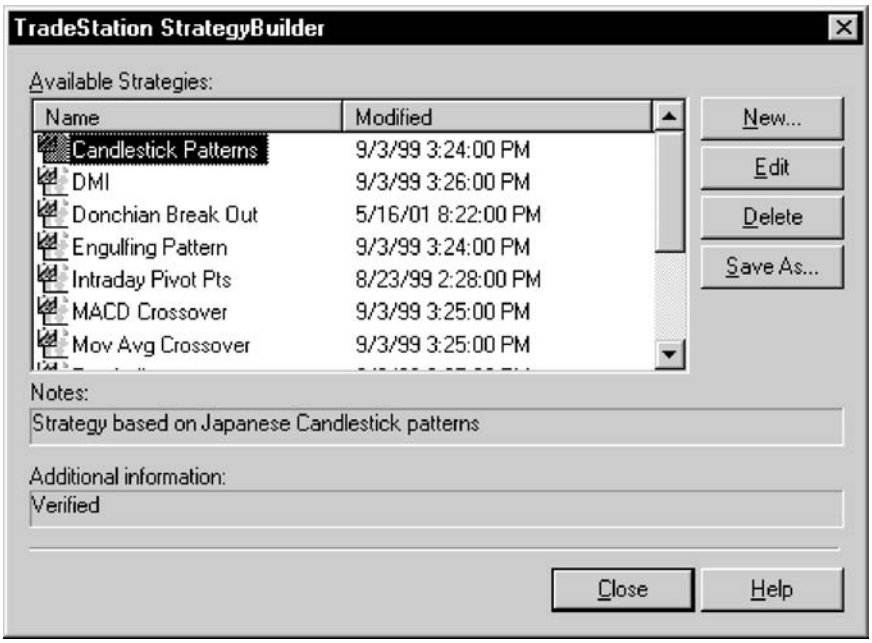

Figure 1.2 TradeStation 2000i—StrategyBuilder

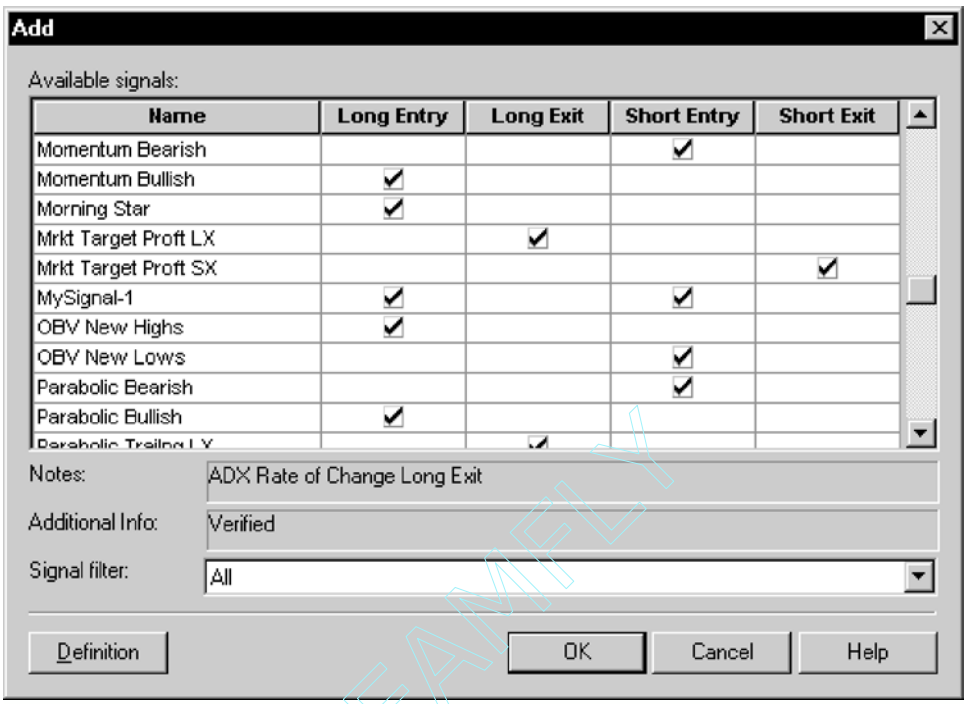

Figure 1.3 TradeStation StrategyBuilder—Add

This dialog is informing us that there are two input variables for our signal. You can change the inputs now, but let's wait until later. If you did want to change the input values, you would simply select the Name of the input and edit the Value. Right now, simply click the Next button. The Pyramiding dialog box opens and asks if you would like to add positions in the same direction. Adding positions in the same direction occurs when our entry logic issues another buy signal and we are already long. We know that we will buy at the highest high of the past 40 days. This dialog is asking if we would like to continue adding positions at each subsequent 40 day high. If the market is trending, a new 40 day high could be made several times in succession. In this book, we will almost always only take one position per trade signal.

Click on the Next button. The *Position Information* dialog now opens and asks for the maximum number of open entries per position and the maximum number of contracts/shares per position. Maximum number of open entries per position limits the number of positions you can add as a result of pyramiding. Maximum number of contracts/shares per position limits the number of total contracts/shares that can be put on per position. The dialog box also asks if you would like it to send a notification to the Tracking Center when our strategy generates a new order. For now, let's accept the default values for the

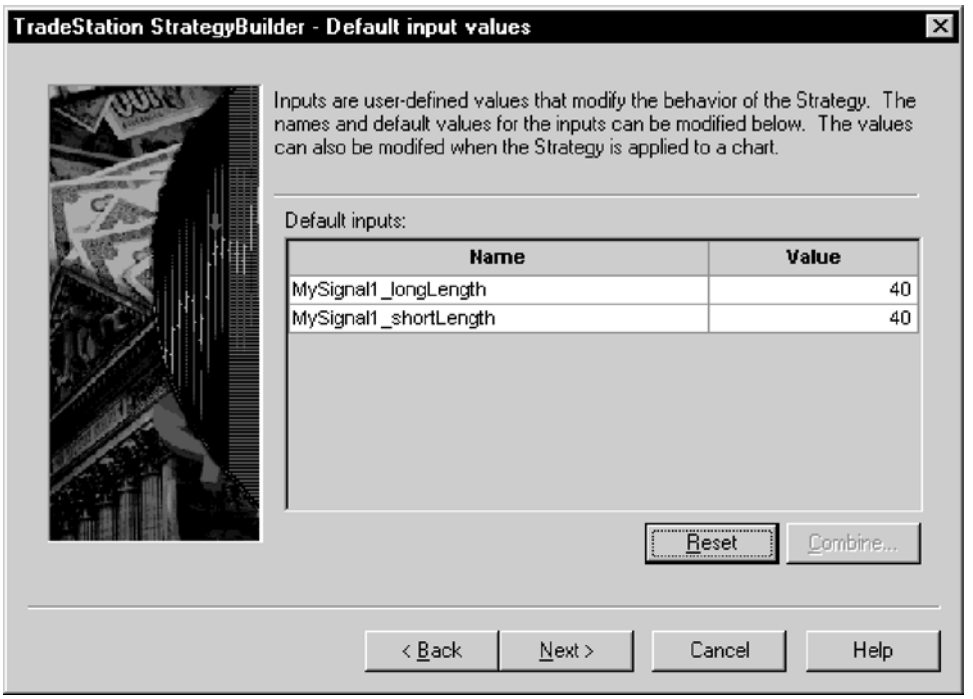

Figure 1.4 TradeStation StrategyBuilder—Input Values

first two fields and make sure the box asking to send a notification to the Tracking Center is checked and then click Next. The *Data Referencing* dialog opens and asks for the maximum number of bars study will reference. Our strategy only needs forty days of data to generate a signal. Remember, we are looking back forty days to find the highest high and lowest low. Always keep in mind how much data the strategy that you are working on requires and make sure that you tell the computer, via this dialog box, that number. Make sure there are 40 or more in the field and click finish. Congratulations, again! You have just created your first strategy. Seems like a lot of work doesn't it? TradeStation is just making sure that all the parameters are correct before testing a signal. Most of the time, these parameters don't change and you can simply click Next, Next, Next, Next, and finally, Finish without paying much attention. Okay, now let's apply our strategy. This book assumes the reader knows how to create daily and intraday charts in TradeStation. Create a daily bar chart of a continuous Japanese Yen contract going back 500 days. When this chart has been plotted, go under the Insert menu and select Strategy. A dialog box titled *Insert Analysis Technique* should open. Click on the Strategy tab and select MyStrategy-1 from the list of available strategies and click on OK. Another dialog box titled *Format Strategy: MyStrategy-1* appears. Click on the Inputs tab. You will see the two inputs that we have coded in our

Donchian Break Out signal. By using the following line in our code, we have given the user of the strategy, be it ourselves or someone else, the ability to change the longLength and shortLength inputs of MySignal-1.

```
Inputs: longLength(40), shortLength(40);
```

You do not need to change the code to change the system from a 40- to a 50-day Donchian Break Out strategy. These inputs can be edited at anytime; before the analysis technique is inserted or after. You will notice that the values of these inputs default to those values that we had initiated when we programmed the signal in the PowerEditor. If you change these inputs from this dialog, they do not permanently change; they will only change during this session. If you want to change the inputs permanently, then change the value and click on the Set default button. If you do change these inputs, always make sure you change the *maximum number of bars the analysis technique will reference* parameter. You can do this by selecting the Properties tab in our dialog box and changing the parameter. Let's take a look at the Properties dialog. Click on the Properties tab. Your dialog window should change and look like the one in Figure 1.5.

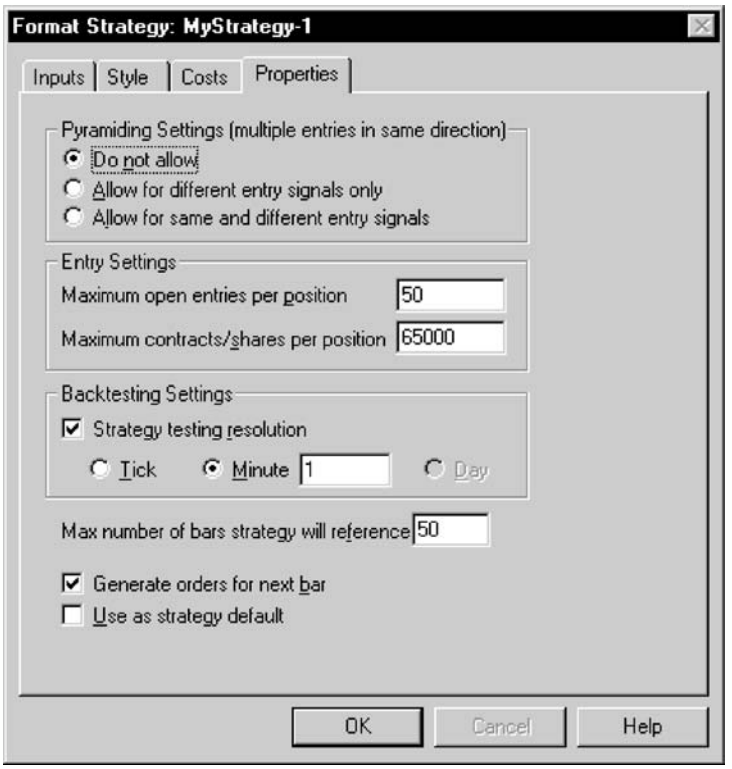

Figure 1.5 Format Strategy—TradeStation 2000i

This dialog box allows the user to observe and change the current properties for MyStrategy-1. We initialized these properties when we first created the Strategy with the StrategyBuilder. You should be familiar with all of the property parameters except for the back testing resolution. TradeStation enables you to specify the resolution or data compression to use for back testing your trading strategy. When you create a chart using data that has already been collected, TradeStation must make certain assumptions about price movement. If you are back testing on daily bars, TradeStation does not know when the high or the low of the day was made. In some strategies, this information may be important. TradeStation calculates the chronological order of the high and low by using a formula. This formula is not always accurate and may lead to inaccuracies. (This concept will be discussed in further depth in Chapter 6, but for right now, let's ignore this option.) Click on the Costs tab and you will be presented with the Commission and Slippage $ values deducted for each trade. We all know what commission is. *Slippage* is the dollar value expected when the actual fill price minus the calculated fill price is calculated on each trade. Slippage is either positive or negative; it is positive when you get in at a better price and negative when you get in at a worse price. Most of the time, slippage is negative. These costs can be charged on per contract/share basis or on a per transaction basis. If you trade 300 shares of AOL, you can have TradeStation charge a commission/slippage on each share or for the entire trade. In addition, you will see the number of contracts/shares your strategy will assume with each new trade signal. If you select Fixed Unit, then this will be the fixed number of contracts/shares that will be traded throughout an historic back test. If you choose Dollars per Transaction and type in a value in the associated text box, the number of shares or contracts will be calculated by dividing the input amount by the price of the stock or by the margin of the futures contract. Go ahead and accept all of the default values by clicking the OK button.

Since MyStrategy-1 is a pure stop and reverse system, you will probably get a *New Open Position* dialog box that states the Market position has changed for JY. Let's close this dialog box by clicking on the Close button. When this dialog window disappears, there should be one underneath it titled *New Active Order*. This dialog box lets you know that an order needs to be placed today. It will either say Buy 1 at a certain price stop or Sell 1 at a certain price stop. We need to learn more about the Tracking Center, so click on the Go to Tracking Center button. You may get another dialog box that states: *No Open Tracking Center windows were found. Would you like to create one?* Go ahead and click Yes. Your Tracking Center should look similar to one in Figure 1.6.

Click on the Open Positions tab and you will see the symbol we are currently testing, current position, entry price, entry time, open profit, and various other statistics. Click on the Active Orders tab and you should see a LE (Long Entry) order to Buy 1 at a certain price stop and a SX (Short Exit) order

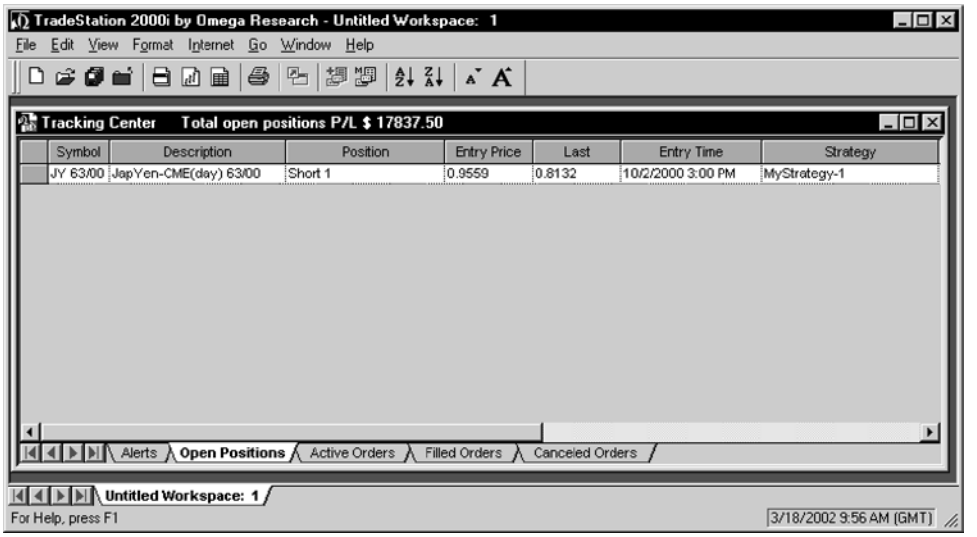

Figure 1.6 Tracking Center—TradeStation 2000i

to Sell 1 at a certain price stop, if you are currently short. If you are currently Long, you would see a SE (Short Entry) order to Sell 1 at a certain price stop and a LX (Long Exit) order to Buy 1 at a certain price stop. In real life order placement, you would simply place a single order to Buy/Sell 2 at whatever price that was issued on a stop. Reduce this window and the chart of the Japanese Yen with buy and sell signals should now be the only window on the screen. If you like, you can go under the View menu and select Strategy Performance Report and look at how well the system performed over the test period. We will go much further in detail concerning the reports that TradeStation creates for the analysis of Trading Strategies in Chapter 5.

TradeStation 6.0

PowerEditor

First off, TradeStation 6.0 must be running and online. Now, launch the PowerEditor and create a new EasyLanguage document by going under the File menu and selecting New. The PowerEditor can also be accessed through the useful Shortcut Bar. Go under the View menu and make sure the Shortcut bar is check marked. Once the bar is open, click on the EasyLanguage bar. All of the options that are available to EasyLanguage documents are now at you disposal. Figure 1.7 illustrates the Shortcut bar.

If you selected New from the File menu, a dialog box similar to the one in Figure 1.8 should now be on your screen.

Figure 1.7 TradeStation 6.0 Shortcut Bar

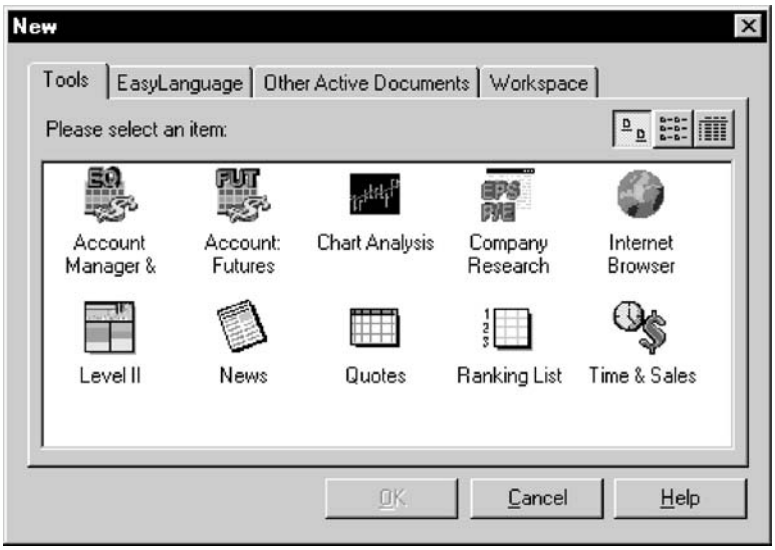

Figure 1.8 New Dialog Box—TradeStation 6.0

After this dialog opens, click on the EasyLanguage tab and select the icon titled *Strategy* and click OK. If you created a new EasyLanguage document from the Shortcut bar, the preceding dialog box is skipped. We will ignore the other tabs in this dialog box at this time. Another dialog box titled *New Strategy* will open and ask for a name and notes about the strategy that you are creating. In the Name field go ahead and type "MyStrategy-1" and in the Notes field type "Donchian Break Out". In the Select Template field, leave it at none and then click OK. A window titled *TradeStation EasyLanguage PowerEditor—MyStrategy-1: Strategy* should open. You have successfully launched PowerEditor. This is your clean sheet of paper on which to type your wonderful ideas. Before we do some programming, let's briefly take a look at some of the menus. Now that PowerEditor is a component of TradeStation 6.0, the two programs that were once separated are now combined and share the same menus. We will only discuss the pertinent menus and menu items that apply to the PowerEditor and EasyLanguage at this time.

Table 1.5 details the selections that are available under the File menu.

Table 1.6 details the selections under the Edit menu.

Table 1.7 details the selections under the View menu.

Table 1.8 details the selections under the Tool menu.

Table 1.5
File Menu

Menu Item	Action
New	Creates many different things—one of which is a blank analysis technique.
Open EasyLanguage Document	Opens an existing analysis technique.
Close Window	Closes window.
Save EasyLanguage Document	Saves the current analysis technique.
Save EasyLanguage Document As	Allows renaming of the current analysis technique.
SaveAs Template	Saves the current analysis technique as a template for future use.
Import/Export EasyLanguage	Imports existing analysis techniques from other sources or exports analysis techniques to other sources. (Note: You can import from previous versions, but you can't export to previous versions.)
Page Setup	Allows changing of the page setup.
Print	Prints the code.
Print Preview	Shows what will be printed.
Exit	Exits out of TradeStation platform.

Table 1.6
Edit Menu

Menu Item	Action
Undo	Undoes the last action.
Redo	Redoes the last action.
Cut	Cuts the selected text.
Copy	Copies the selected text.
Paste	Pastes the selected text.
Clear	Clears the selected text.
Select All	Selects all text in the analysis technique.
Find	Finds a specific string of text.
Find Next	Finds the next occurrence of the specified string of text.
Replace	Replaces a string of text with another string of text.
Find in Files	Powerful multifile search tool.

Table 1.7
View Menu

Menu Item	Action
Order Bar	Hides/shows the Order bar
Shortcut Bar	Hides/shows the Shortcut bar
Status Bar	Hides/shows the Status bar
Output Bar	Hides/shows the Output bar. This little window will become important when we begin debugging.
EasyLanguage Preferences	Sets the PowerEditor's font, background color, and syntax coloring.

Table 1.8
Tool Menu

Menu Item	Action
Verify	Verifies the analysis technique. Checks for syntax errors in code.
Protect Document	Protects code with password.
Easy Language Dictionary	Inserts EasyLanguage Components into an analysis technique.

A Simple Program

Now that we are familiar with some of the menus and their functions, let's code a simple program. We don't need to know everything about the selections in the menus to start coding. Type the following text exactly as you see it here:

```
Inputs: longLength(40), shortLength(40);

Buy tomorrow at Highest(High,longLength) stop;
Sell Short tomorrow at Lowest(Low,shortLength) stop;
```

After you have typed this in, go under the Tools menu and select Verify or press F3. You should get a small dialog box that first says *Verifying* and then *Verification Successful*. If you get an error, simply check your code for any typos and Verify again. Congratulations, you have just written an analysis technique in the form of a strategy! Now let's break each line of code down so that we can fully understand what is going on.

```
Inputs: longLength(40), shortLength(40);
```

By typing this line you have created two constant variables of the numeric data type. These two variables, longLength and shortLength, have been initiated with the value of 40. These variables cannot be changed anywhere in the body of the analysis technique. They can be changed from the user interface of this signal or in the program header. These will be discussed later in this chapter.

```
Buy tomorrow at Highest(High,longLength) stop;
```

This line instructs the computer to place a buy stop tomorrow at the highest high of the last forty days. Highest is a function call. Functions are subprograms that are designed for a specific purpose and return a specific value. To communicate with a function, you must give it the information it needs. An automatic teller machine is like a function; you must give it your PIN before it will give you any money. In the case of the Highest function, it needs two bits of information: what to look at and how far to look back. We are instructing this function to look at the highs of the last 40 bars and return the highest of those highs. When an order is instructed through EasyLanguage, you must tell the computer the type of order. In this case, we are using a stop order. Orders that are accepted by EasyLanguage are:

- *Stop*. Requires a price and is placed above the market to buy and below the market to sell.
- *Limit*. Requires a price and is placed below the market to buy and above the market to sell.
- *Market*. Buys/sells at the current market price.

So, if the market trades at a level that is equal to or greater than the highest high of the past forty days, the signal would enter long at the stop price.

```
Sell Short tomorrow at Lowest(Low,shortLength) stop;
```

The instructions for entering a short position are simply the opposite for entering a long position.

For those of you who are moving from TradeStation 2000i to 6.0, you will be pleasantly surprised by the elimination of the StrategyBuilder. Version 6.0 creates strategies on the fly by accepting default values for the strategy's properties. For those of you who are not familiar with the StrategyBuilder, it was an additional component that required the user to click through several dialog boxes and change property values before one could create a strategy. Most of the time the default values were sufficient and this was an act of futility. Version 6.0 allows the strategy properties to be changed when needed.

This book assumes the reader knows how to create daily and intraday charts in TradeStation. If you are not sure how to do this, we refer you to your TradeStation manual. Create a daily bar chart of the Japanese Yen going back 500 or more days. When this chart has been plotted, go under the Insert menu and select Strategy. A dialog box titled *Insert Analysis Techniques and Strategies* should open and look similar to the one in Figure 1.9.

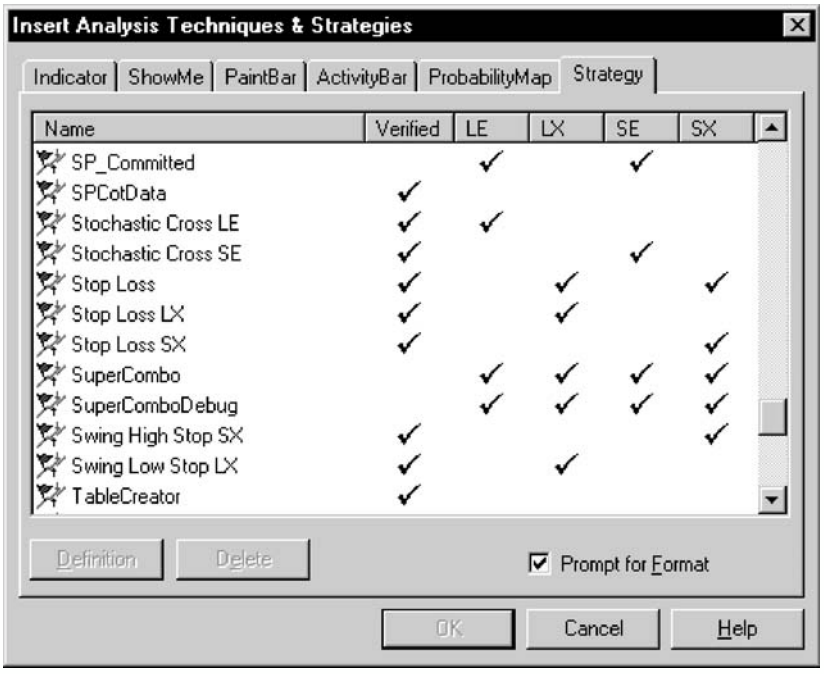

Figure 1.9 Insert Analysis Techniques and Strategies

This dialog box is very informative: It lists the different strategies and also informs the user if the strategy has been verified and if the strategy has a long entry, short entry, long exit, and short exit. Scroll through the list until you find MyStrategy-1. You will notice that the boxes under the Long Entry and Short Entry column headings are checked, but the boxes underneath the Long Exit and Short Exit are not. This tells us that the system only enters the market; it is never flat—the system is either long or short. Long positions are liquidated when a short position is initiated and vice versa. Our simple Donchian Break Out is a pure stop and reverse system. You will also notice a small check box titled *Prompt for Format*. Make sure this box is checked and then select MyStrategy-1 from the list of available strategies and click on OK. Another dialog box titled *Format Strategy* appears and should look similar to Figure 1.10.

Click on the Inputs button. You will see the two inputs that we have coded in our Donchian Break Out strategy. By using the following line of code, we have given the user of the strategy, be it ourselves or someone else, the ability to change the longLength and shortLength inputs of MyStrategy-1.

```
Inputs: longLength(40), shortLength(40);
```


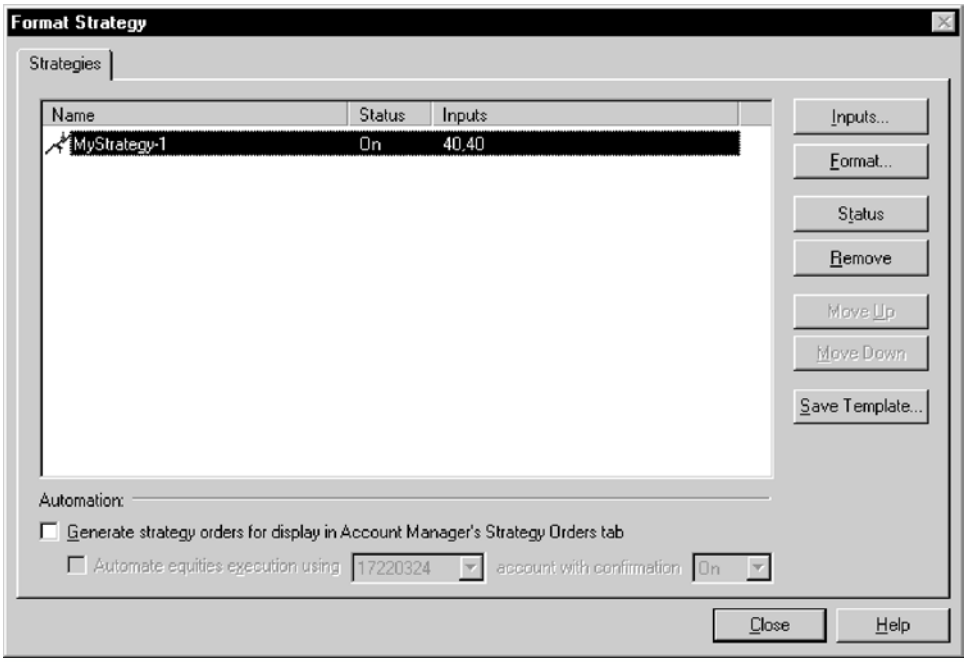

Figure 1.10 Format Strategy

You do not need to change the code to change the system from a 40- to a 50-day Donchian Break Out. These inputs can be edited at anytime; before the analysis technique is inserted or after. You will notice that the values of these inputs default to those values that we had initiated. If you change these inputs from this dialog, they do not permanently change; they will only change during this session. You can permanently change the default input values for MyStrategy-1 by changing the values and then clicking on the Set Default button. If you do change these inputs, always make sure you change the *maximum number of bars the analysis technique will reference* parameter. We will show you how to reference this parameter when we discuss the Format dialog box. You will notice that there are several other buttons in this dialog box, however we will ignore these for now and accept our Inputs as they are by clicking on the OK button. Click on the Format button. A dialog box should open titled *Format Strategy*. It should look like Figure 1.11.

You will notice several different parameters that TradeStation takes into account when testing a strategy. Let's examine each option and its purpose.

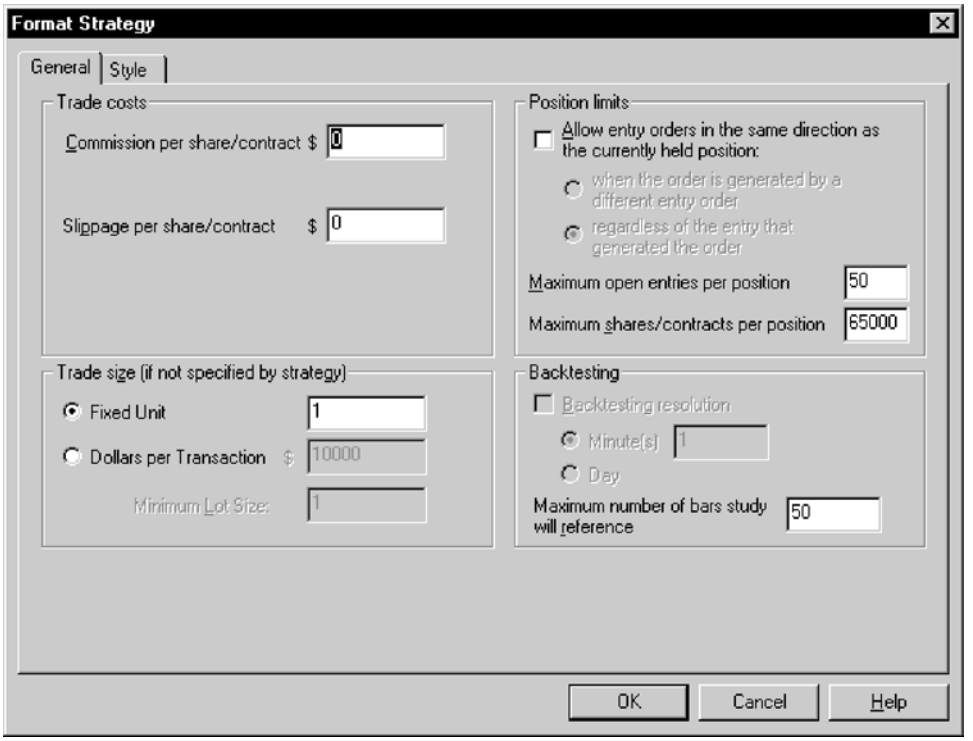

Figure 1.11 Format Strategy—After Selecting Format

Commission per share/contract $: The dollar value that will be deducted from each trade as a commission charge. Let's set this to $0.

Slippage per share/contract $: The dollar value that you expect the strategy will be slipped on each trade. Slippage is the actual fill price minus the calculated fill price. Slippage is either positive or negative; it is positive when you get in at a better price and negative when you get in at a worse price. Most of the time, the slippage is negative. Let's set this to $0.

Trade size (if not specified by strategy)

Fixed Unit: The number of shares or contracts that are put on at the initiation of a trade. Change this value to 1, if it isn't already so.

Dollars per Transaction: The fixed dollar value used to determine the number of shares/contracts. Trade size is calculated by dividing the price of the instrument into this value.

Go ahead and click OK and you return to the *Format Strategy* dialog box. Before we close this dialog and proceed with the application of the strategy to the chart, make sure the *Generate strategy orders for display in Account Manager's Strategy Order Tab* check box is checked. Since MyStrategy-1 is a pure stop and reverse system, you will get a *Strategy New Order* or a *Strategy Active Order* dialog box that states that you should buy or sell short at a certain price. If one of these dialog boxes does indeed open, just close it by clicking on the Close button.

We need to learn more about the Account Manager and Strategy Tracking. Go to the File menu and select New. A dialog window should open and look similar to the one that we saw when we created a new strategy. However, this time, click on the Tools tab. Click on the Account Manager and Strategy Testing icon. We could have accomplished this by using the Shortcut bar. Instead of clicking on the EasyLanguage bar, we would have clicked on Tools and then proceeded to click on the Account Manager and Strategy Testing icon. We have simply fallen in love with the Shortcut bar. Once you click the icon, a window like the one in Figure 1.12 will open.

This window keeps track of all real orders and positions (if you have an actual trading account with TradeStation securities) and all simulated orders and positions. Since we are working with simulated trades that were generated by our strategy, click on the Strategy Positions tab. Your window will change and should look like the one in Figure 1.13.

This spreadsheet shows the symbol we are currently testing, current strategy position, entry price, entry time, and various other statistics. Click on the Strategy Orders tab and your window will change to look like the one in Figure 1.14.

Since our strategy is in the market all of the time, you will have two rows (possibly more if you are currently tracking more than one system) of

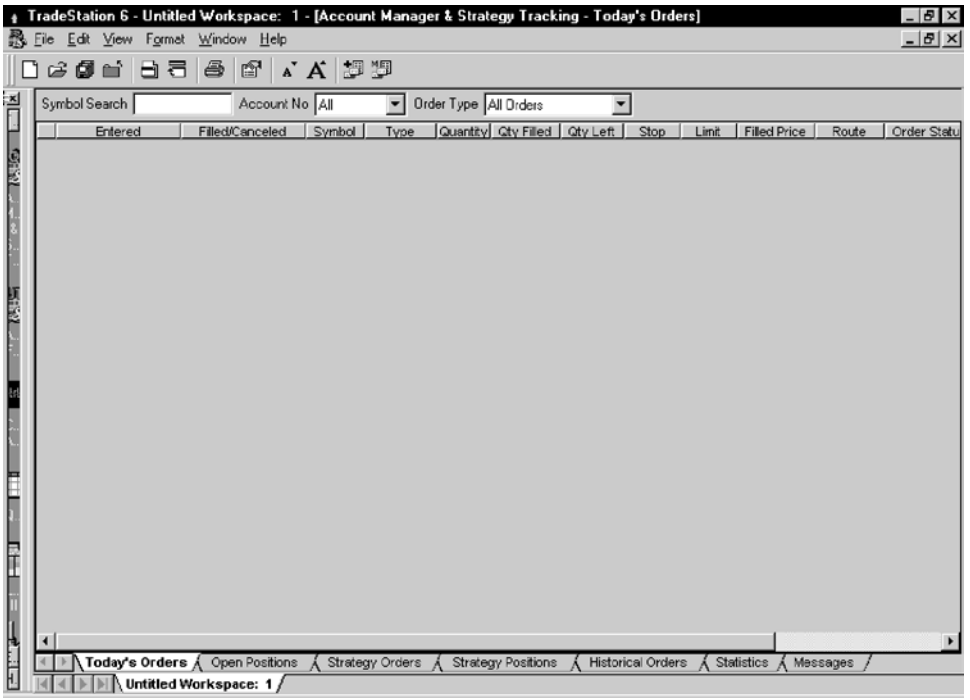

Figure 1.12 Account Manager and Strategy Tracking—Today's Orders

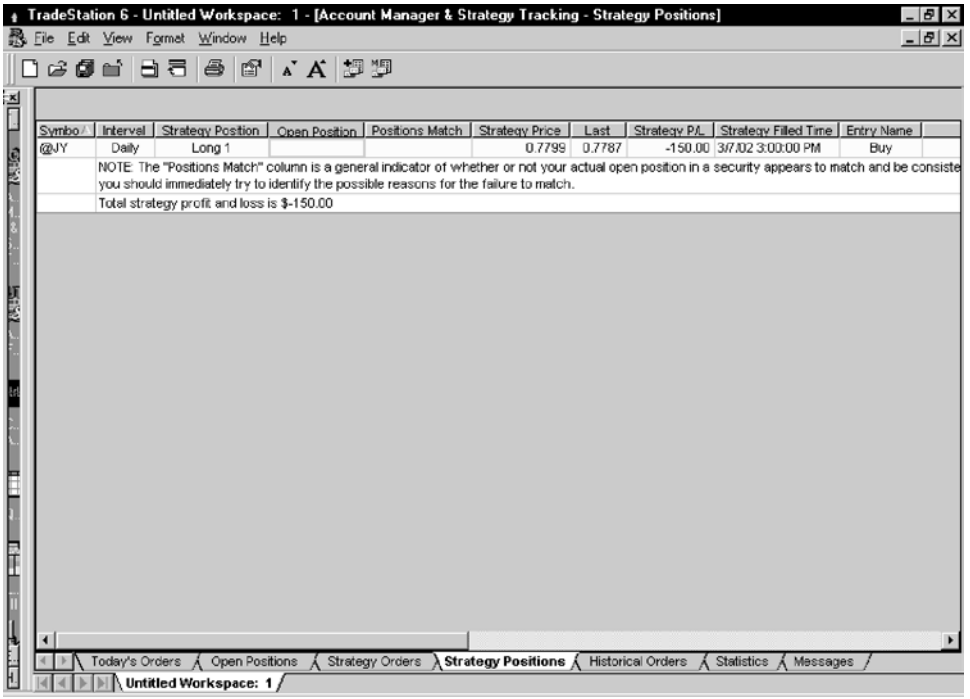

Figure 1.13 Account Manager and Strategy Tracking—Strategy Positions

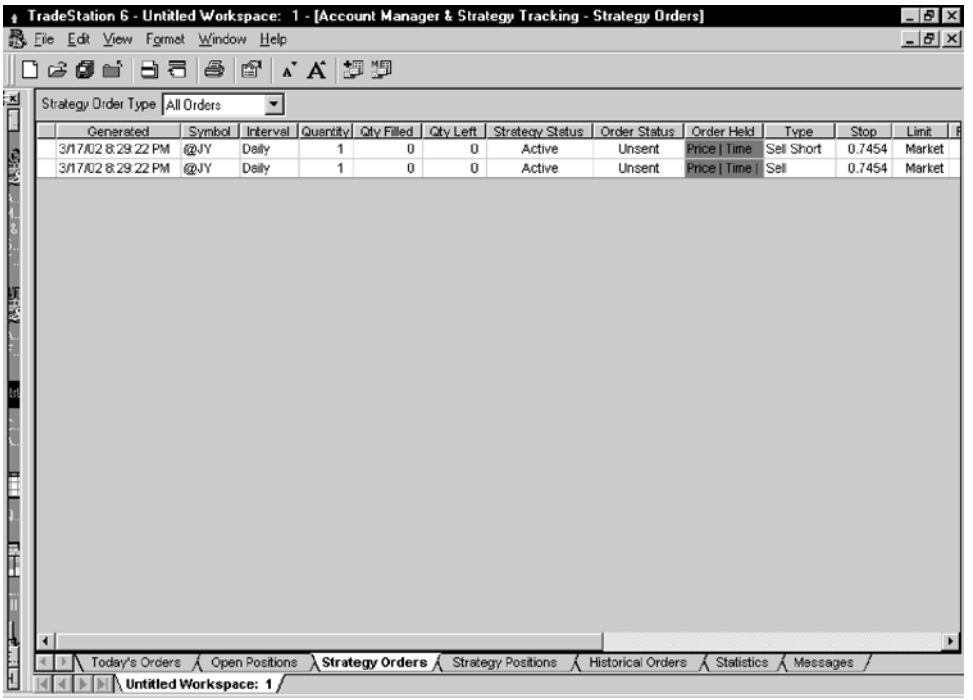

Figure 1.14 Account Manager and Strategy Tracking—Strategy Orders

information. The top row will be the order to initiate a new position and the second row will be the order to cover the existing position. If the order is a stop order, the stop price will be located under the Stop header. If it is a limit order, the limit price will be under the Limit header. If today's market action causes a fill to occur, the fill price will be under the Filled header. You will notice other tabs in the Account Manager window. These tabs are used only if you have an account set up at TradeStation's brokerage company. Basically, these other tabs give the same information as the Strategy tabs, but with real execution statistics. You can use TradeStation 6.0 without a trading account, but you will need to keep track of your real live positions and fills yourself. This in no way takes away from the back testing and research capabilities of this product.

Reduce this window and the chart of the Japanese Yen with buy and sell signals will now be the only window on the screen. If you like, you can go under the View menu and select Strategy Performance Report and look at how well the system performed over the test period. We will go in much further detail concerning the reports that TradeStation creates for the analysis of Trading Strategies in Chapter 5.

CONCLUSIONS

The objective of this chapter is to introduce the fundamentals necessary to become productive EasyLanguage programmers. We discussed data types, expressions, and statements. We reviewed how to declare variables and inputs, call built-in functions, and verify (compile) our code. In addition, we can create a strategy from a simple signal and insert that strategy into a chart. In the next chapter, we will build upon this foundation and create much more complex programs (analysis techniques in EasyLanguage vernacular).

2

EasyLanguage Program Structure

STRUCTURED PROGRAMMING

Structured programming was introduced in the early 1970s. This concept stressed breaking a program down into manageable modules and then connecting those modules together into a coherent and logical flow of instructions. We know that the readers of this book will probably not become professional programmers, but structured programming is necessary for the accurate transfer of ideas into action. Any time you add structure to anything, you are always better off. EasyLanguage was developed as an easy-to-learn language for traders. It was not intended to produce professional programmers, but instead to get traders to write some simple programs. Through our years of programming with EasyLanguage, we have discovered that most analysis techniques programs can be broken down into three different modules: a program header, a calculation module, and an order placement module. This modularization is not necessary to program analysis techniques. In fact, since EasyLanguage has so many shortcuts, many programmers prefer quick and dirty "spaghetti" code (code that is as disorganized as a plate of spaghetti). This quick programming is fine for simple brainstorming, but when your analysis techniques become complicated, structured programming will save you time in the long run. It is also necessary for debugging purposes and you will be doing some debugging.

PROGRAM HEADER

The header of an EasyLanguage program is the portal for communication between the internals of the program and the outside world. As in Chapter 1, the Input statement allows the user of the analysis technique to modify particular parameters without having to rewrite and reverify the technique. The header also declares and initializes variables that will be used later in the analysis technique. This is the starting point of our structured program. The program header is an excellent place to describe the objective of our programming through the use of comments. The following is a good example of a structured program heading:

```
{MyRsiSystem – trading strategy by George Pruitt 09/24/2001

Version 1.00 initial testing of my idea
Version 1.01 added RSI retracement component
Version 1.02 added trailing stop mechanism
Version 1.03 changed the over bought/sold parameter

This trading signal is designed to buy when the RSI has dipped into the oversold
territory and sells when the RSI has risen into overbought territory. Once a
position is initiated an initial money management stop is invoked and then a
trailing stop takes over after a certain threshold of profit is attained.}

Inputs: rsiLength(14), overSold(40), overBought(60), moneyManStop(1000);
Vars: myRsiVal(0),longProtStop(0),shortProtStop(99999),obCount(0),osCount(0);
Vars: longProfitStop(99999),shortProfitStop(0),takeProfitStop(2000);
```

The header starts out with the name of the analysis technique, the author, and the date. Notice that remarks or comments are sandwiched between curly brackets ({}). The curly brackets inform the compiler to ignore these statements. These brackets can also be used to "comment out" code that you may want to keep in your program, but don't want to necessarily use at this time. Keeping track of changes is always important, especially when designing trading techniques. There have been a number of times that we have come up with a good basic idea, only to forget and lose it after we have changed it umpteen times. If we had only kept track of the revisions, we could get back to the original idea. In this program header, we have recorded the additions and changes by applying different version numbers. This is similar to the method that large software companies incorporate into their own software development. A simple explanation of the analysis technique follows. After you have developed many different strategies, indicators, and so on, it becomes difficult to differentiate your ideas by simply looking at the name of the analysis technique. By putting a brief description of the analysis technique at the top of your program,

you can quickly figure out what this particular program is attempting to do. This is also helpful if you are going to share the code with others. Most programmers hate to put comments in their code; they feel that it is a big waste of time. Most traders are programming for themselves and do not need to share their code with others. This may be the case, but we all need reminders and explanations of our ideas, especially after some time has elapsed. This type of program heading is appropriate for any analysis technique; be it a Strategy, PaintBar, Indicator, or ShowMe.

Since we are discussing readability, you should always save your analysis techniques with good, self-descriptive names. Saving an idea under the name of MyTradingSys1 will create confusion. After a couple of weeks of coding, we guarantee you will forget the main theme behind MyTradingSys1. Reusing code is one of the main reasons for building a library and if you don't label your ideas correctly, you will waste time searching for them. It would be like trying to find a book in a large public library without the card catalog. Some good files names that give a general idea of what code is designed to do are: MyRSIsystem, BreakOutSys, ChannelStrat, MyStochIdea, or BollingerTrader.

The Inputs and Vars declaration statements come next. Notice how the inputs and variables are named; they all are self-descriptive.

CALCULATION MODULE: MYRSISYSTEM

This module is where the variables and inputs are put to work. When we talk about modules, we mean a grouping of similar code. A module isn't a function or a subroutine; they could be, but in our examples, modules are just separate areas of code. By separating the code, we can easily read and understand the thought process that went into the coding. Take a look at the following snippet of code from the calculation module of the MyRSIsystem.

```
myRsiVal = RSI(close,rsiLength);
if (myRsiVal > overBought and myRsiVal[1] < overBought) then
begin
       obCount = obCount + 1;
       osCount = 0;
end;
if (myRsiVal < overSold and myRsiVal[1] > overSold) then
begin
       osCount = osCount + 1;
       obCount = 0;
end;
```

You may not fully understand what is going on at this point. That's okay. There are a few important ideas that are introduced in this code that we really do need

to understand before we can go on, and we will explain them completely. Before we explain these new concepts, notice how we indented some of the program statements. This is for readability. If a particular statement controls the execution of a line or lines of code, then those lines should be indented so that it is easy to see which statements control which statements. Notice that the lines of code that start with osCount and obCount are indented. The *if-then* statements control the execution of these lines of code. Again, these indentations are not necessary; it just makes your code easier to read and to understand. We will discuss program control structures in Chapter 3.

The statement myRsiVal = RSI(close, rsiLength) calls a built-in EasyLanguage Relative Strength Index (RSI) function. The RSI function returns a value based on your inputs. In this case, we instruct EasyLanguage to calculate the RSI on the past 14 days' closing prices. (We introduced the concepts of functions in Chapter 1.)

A function is called by simply typing the function name (in this case, RSI) and passing the necessary inputs or parameters to it. Passing values to functions is quite simple. First, type the name of the function and a left parenthesis, and the necessary parameters separated by commas and a right parenthesis. The first parameter we pass to the RSI function informs the function to use the closing prices. The second parameter that we pass informs the function to use the past 14 days. Before you call a function, you must know the number of parameters, the type of parameters, and the exact order of the parameters that the function is expecting. (There is a list of frequently used functions and their parameter lists in Appendix A.) If you do not pass the correct parameters to a function, you will either end up with a syntax or logical error. A syntax error is generated when the compiler does not understand what it is being told. The following function call will generate a syntax error: myRsiVal = RSI(Close); . The compiler will inform you that more inputs are expected.

In Chapter 1, we discussed the reserved words that you can use to gain access to the bar chart data (*Open*, *High*, *Low*, *Close*, etc.), and how we are able to reference historical prices by adding an index into the reserved words. You can do the same thing with your own defined variables. If you want to look at the value of a variable on the previous bar, all you have to do is add the index. In the earlier sample code, notice how the previous bar's RSI value was compared to the overBought value:

```
myRsiVal[1] < overBought.
```

If you wanted to look at the RSI value of five bars back, you would type myRsiVal[5]. This is a powerful feature of EasyLanguage; it keeps track of the previous values of your variables so that you can access them anytime you need to. There is one limitation; EasyLanguage will only remember the number of bars that is specified by the MaxBarsBack setting. If you set the MaxBarsBack

setting to 10, and you try to reference myRsiVal[11] (11 days back), you will not receive an accurate result.

The following code shows how we have modularized the order placement code:

```
{Order Placement Module}

if(obCount = 2) then SellShort tomorrow at open; {We have entered OB twice}
if(osCount = 2) then Buy tomorrow at open; {We have entered OS twice}

if(marketPosition = 1) then
begin
       Sell ("longLoss")next bar at longProtStop on a stop;
       Sell ("longProfit")next bar at longProfitStop on a limit;
end;
if(marketPosition = -1) then
begin
       BuyToCover ("shortLoss")next bar at shortProtStop on a stop;
       BuyToCover ("shortProfit")next bar at shortProfitStop on a limit;
end;
```

Notice how the entry orders and exit order calculations are placed in their respective groups. This makes it easier to go to the exact location in the code without having to hunt and search. Again, don't worry about what we are trying to accomplish in this code, just try to teach yourself how to program and think in modules. The order of these buy and sell orders is not important, because TradeStation evaluates the orders simultaneously. Many times you may have more than one order working. In our example, the number of working orders depends on our current position. If we are long, then we have a short entry order, and two long exit orders. This is the same for a short position. If we are flat, then we have only one entry order working based on our osCount variable. To enter long, our osCount variable must be equal to 2 and to enter short, our obCount must be equal to 2 also. TradeStation only executes the order that is closest to the market.

Now let's look at MyRsiSystem in its entirety:

```
{MyRsiSystem - trading strategy by George Pruitt 09/24/2001

Version 1.00 initial testing of my idea
Version 1.01 added RSI retracement component
Version 1.02 added profit objective mechanism
Version 1.03 changed the over bought/sold parameter

This trading signal is designed to buy when the RSI has double dipped into
the oversold territory and sells when the RSI has doubly risen into
```

```
overbought territory. Once a position is initiated an initial money
management stop and profit objective stop is invoked.}

Inputs: rsiLength(14), overSold(40), overBought(60), moneyManStop(1000);
Vars: myRsiVal(0),longProtStop(0),shortProtStop(99999),obCount(0),osCount(0);
Vars: longProfitStop(99999),shortProfitStop(0),takeProfitStop(2000);
{Calculation Module}

myRsiVal = RSI(close,rsiLength);
if(myRsiVal > overBought and myRsiVal[1] < overBought) then
begin
       obCount = obCount + 1;
       osCount = 0; {Reset the OS counter since we are OB}

end;
if(myRsiVal < overSold and myRsiVal[1] > overSold) then
begin
       osCount = osCount + 1;
       obCount = 0; {Rest the OB counter since we are OS}
end;

if(marketPosition = 1) then
begin
       longProtStop = entryprice – (moneyManStop/PointValue/PriceScale);
       longProfitStop = entryprice + (takeProfitStop/PointValue/PriceScale);
       osCount = 0; {Since we are long – reset the OS counter}
end;
if(marketPosition = –1) then
begin
       shortProtStop = entryprice + (moneyManStop/PointValue/PriceScale);
       shortProfitStop = entryprice – (takeProfitStop/PointValue/PriceScale);
       obCount = 0; {Since we are short – reset the OB counter}
end;

{Order Placement Module}

if(obCount = 2) then Sell Short tomorrow at open; {We have entered OB twice}
if(osCount = 2) then Buy tomorrow at open; {We have entered OS twice}

if(marketPosition = 1) then
begin
       Sell ("longLoss")next bar at longProtStop on a stop;
       Sell ("longProfit")next bar at longProfitStop on a limit;
end;
if(marketPosition = –1) then
begin
       BuyToCover ("shortLoss")next bar at shortProtStop on a stop;
       BuyToCover ("shortProfit")next bar at shortProfitStop on a limit;
end;
```

It is easy to see the different "modules." Also, notice how we have used comments to help explain the purpose of different statements. As a sidebar, the EasyLanguage compiler is not case sensitive: obcount is the same as obCOUNT, ObcOuNt, OBCOUNT, and so on. This applies to all variables, keywords, function names, and inputs. We have a certain nomenclature when it comes to using upper and lowercase letters that we follow when we create variables. Most of the time we type in lowercase and only use uppercase at the beginning of functions names and variables that deal with the data: *Open*, *High*, *Low*, *Close*, *Volume*, *Date*, and *Time*. As explained in Chapter 1, we also use uppercase letters at the beginning of a new syllable (other than the first) in our own variable names. We do this to differentiate our variables from built-in keywords and functions. Also, if you type this system or import it into the PowerEditor, you will notice that some of the words are different colors. This is a fantastic feature of the PowerEditor; syntax coloring can save you many hours of debugging and research time. EasyLanguage has a vast library of keywords and functions and it would be impossible to memorize them all. In the default settings of the PowerEditor, reserved words and function names have a different color than normal text. With syntax coloring, you can guess at a name and if it is a valid reserved word or function name, it will turn a different color. Many times I will need to call a function, but I can't remember the exact name, so I will type what I think the function name should be and if it doesn't turn a different color than the normal text, I try again. For some reason, I can never remember the exact name for the average true range function. I always type "averageTrueRange" and it never turns a different color. I then try "avgTrueRange" and it does turn and I know that I have the correct name. The EasyLanguage preferences dialog box allows you to customize the color of comments, reserved words, functions, skip words, quote fields, and string text. Set these types of words to a different color than normal text. Now, let's look at the same code in a nonmodular format and without comments. You determine which code is easier to interpret.

```
{MyRsiSystem - trading strategy by George Pruitt 09/24/2001}
{Spaghetti Code}

Inputs: rsiLength(14), overSold(40), overBought(60), moneyManStop(1000);
Vars: myRsiVal(0),longProtStop(0),shortProtStop(99999),obCount(0),osCount(0);
Vars: longProfitStop(99999),shortProfitStop(0),takeProfitStop(2000);

myRsiVal = RSI(close, rsiLength);

if(osCount = 2) then buy tomorrow at open;

if(marketPosition = 1) then
begin
       longProtStop = entryprice – (moneyManStop/PointValue/PriceScale);
```

```
        longProfitStop = entryprice + (takeProfitStop/PointValue/PriceScale);
        osCount = 0;
end;
if(obCount = 2) then sell short tomorrow at open;
if(marketPosition = –1) then
begin
        shortProtStop = entryprice + (moneyManStop/PointValue/PriceScale);
        shortProfitStop = entryprice – (takeProfitStop/PointValue/PriceScale);
        obCount = 0;
end;
if(marketPosition = 1) then
begin
        Sell ("longLoss")next bar at longProtStop on a stop;
        Sell ("longProfit")next bar at longProfitStop on a limit;
end;
if(marketPosition = –1) then
begin
        BuyToCover ("shortLoss")next bar at shortProtStop on a stop;
        BuyToCover ("shortProfit")next bar at shortProfitStop on a limit;
end;

if(myRsiVal > overBought and myRsiVal[1] < overBought) then
begin
        obCount = obCount + 1;
        osCount = 0;

end;
if(myRsiVal < overSold and myRsiVal[1] > overSold) then
begin
        osCount = osCount + 1;
        obCount = 0;
end;
```

We think you will agree that the modular version was much clearer. Since the compiler evaluates statements from top to bottom and one line at a time, certain variables may need to be altered before an order is placed. Modularization, in addition to improved readability, adds correct logic flow to your programs. We will use the modular version of this system in Chapter 3 to help explain EasyLanguage programming. By the time we are through, you will fully understand every line of code in this system and be able to use the concepts to build your own analysis techniques from scratch.

CONCLUSIONS

The most important concept learned in this chapter is modular programming. An accurate program (analysis technique) is constructed by using individual

building blocks. In our case, the building blocks are sections or modules of code that are utilized to improve readability and correctness. In addition, we discussed that built-in and user-defined variables can be indexed to look back at previous values. Remember that PowerEditor is user-friendly, is not case sensitive, and recognizes important reserved words and function names through the use of syntax coloring. The next chapter will tear MyRSIsystem apart and introduce and explain some important programming concepts.

3

Program Control Structures

The least complicated programs start at the top of the program block and execute each statement in order and stop after the last statement. A very simple and straightforward strategy is illustrated by our very first strategy from Chapter 1:

```
Inputs: longLength(40), shortLength(40);
Buy tomorrow at Highest(High,longLength) stop;
Sell Short tomorrow at Lowest(Low,shortLength) stop;
```

Most trading ideas can rarely be expressed in such simplistic terms. This strategy does not take into account a protective or trailing stop, profit objective, or any other exit mechanisms. It's not that a simple approach can't work (most of the time they work best), but trading ideas can be complex and involved.

CONDITIONAL BRANCHING WITH IF-THEN

You can make your programs as complex as you need to by using control structures. These structures give programs the ability to react differently under different situations; based on information provided to it, a program can choose between different avenues of logic to follow. In other words, your program must make a decision. Decision processing requires three bits of information: (1) what information is used, (2) how to evaluate the information, and (3) what to do after the decision. This type of programming is called *conditional branching*, because the flow of your program will branch in different directions after

a logical condition is evaluated. Conditional branching is a form of a control structure. By adding conditional branching to the earlier strategy, it has the ability to liquidate a position with a different exit. Let's add the code that will exit a long/short position after five or more trading days if the position is not profitable.

```
Inputs: longLength(40), shortLength(40);
Buy tomorrow at Highest(High,longLength) stop;
Sell Short tomorrow at Lowest(Low,shortLength) stop;

If(marketPosition = 1 and barsSinceEntry(0) > = 5 and Close < entryPrice) then
      Sell("LongLoss5Days) on this bar close;
If(marketPosition = –1 and barsSinceEntry(0) > = 5 and Close > entryPrice) then
      BuyToCover("ShortLoss5Days) on this bar close;
```

This small addition of code has introduced a new concept, reserved word, and function. Before we explain exactly what is going on behind the scenes of this new code, let's learn the concept of the conditional branching control structure. The brain behind conditional branching is the conditional expression, or Boolean expression. A conditional expression is any expression that results in either a true or false condition.

Condition statements consist of the comparison between one or more values. You can have direct comparisons:

```
myProfit > 1200
myRsi <= 20
close[1] > close[2]
```

or you can have comparisons based on calculations:

```
myProfit >= entryPrice + 1200
close[1] > close[2] + (high[2] – low[2])
```

or you can have multiple comparisons:

```
marketPosition = 1 and barsSinceEntry(0) >= 5 and close < entryPrice
(myRsiVal > overBought and myRsiVal[1] < overBought)
```

All conditional expressions reduce down to a left side value that is compared to a right side value using a logical operator. Remembering the operators learned in school:

> Greater than

< Less than

>= Greater than or equal to

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

These operators can be used to compare any two expressions when the two items being compared are compatible. You would not want to compare a string value to a numerical value or a string value to a logical value. Simple expressions are easy to understand. For example, (myRsi > 20) is clearly understood to mean myRsi value is greater than 20. Since we are human and traders (smarter than the average person, right?), most of our thought processes are not this simple. For this reason, EasyLanguage has provided the *or* operator and the *and* operator. Most of the time we will use these operators in the form of logical operators; we will use them for comparison purposes. They can also be used in the form of arithmetic operators, but for our purposes we rarely do this. To fully understand these operators, you must understand the following truth table.

```
Value1      Value2      AND result     OR result
True        True        True           True
True        False       False          True
False       True        False          True
False       False       False          False
```

Here are some examples of how conditional statements are evaluated. Take the following variable assignments:

```
myValue1 = 10;
myValue2 = 7;
myValue3 = 3;
```

Now based on the previous truth table, the following condition statements evaluate to either true or false:

```
myValue1  = 10 and myValue2 = 7                        True
myValue1  = 10 or myValue2 = 4                         True
myValue1  = 10 and myValue3 = 6                        False
myValue1 >= 10 and myValue2 <= 7                       True
myValue1  = 10 and (myValue2 = 6 or myValue2 = 7)      True
```

The last statement may throw you at first. But, remember back in Chapter 1 when we discussed the precedence of operators and how parentheses can change their order. In this example, we have True and (False or True), which reduce down to True and True. We first evaluate the information inside the parentheses and then do the next comparison. Inside the parentheses, we have

False or True, which is True. We then compare True and True and as we all know this is true.

The *if-then* statement is the simplest form of conditional branching. This statement causes the program to execute a single statement or a block of code if a condition is true. When the compiler first encounters an *if-then* statement, it evaluates the information that is provided by the conditional statement. The evaluation produces one of two possible results—true or false. If the statement is true, the line or block of code immediately following the *if-then* is executed. If the result is false, the program skips the line or block entirely. The *if-then* construct has the following syntax (syntax are the rules governing which statements and combinations of statements in a programming language will be acceptable to a compiler for that language):

```
if(conditional statement) then [single statement];
if(conditional statement) then
begin
       [multiple statements]
end;
```

You must always include the word *then* after the *if* conditional statement. The parentheses around the conditional statement(s) are optional, but they do help in clarification. If you want more than one statement to be executed after a conditional branch, you must use the words *begin* and *end*. *Begin* marks the beginning point of the block of code and *end* literally marks the end of the block of code. Semicolons are not placed after the keywords *then* or *begin*. Referring back to the code that we added to MyStrategy-1:

```
If(marketPosition = 1 and barsSinceEntry(0) >= 5 and close < entryPrice) then
      Sell("LongLoss5Days") on this bar close;
If(marketPosition = –1 and barsSinceEntry(0) >= 5 and close > entryPrice) then
      BuyToCover("ShortLoss5Days") on this bar close;
```

Here, the *if-then* control structure liquidates our position if the position is not profitable after five or more trading days. The program tests our position on a daily basis after the fourth trading day and it will liquidate it if a closing price results in negative equity. You may look at the code and ask, "How do we know the number of days we have been in a trade?" Thanks to the vast library of EasyLanguage, we have a built-in function that can tell us this information. The function barsSinceEntry returns the number of days that we have been in a position. Remember, most functions need to be passed some type of information; barsSinceEntry is similar in that it needs to know which position we are talking about. In this particular case, we want the most recent position. By passing a zero to the function, it knows that we need information pertaining to the latest position. If we had passed a 1, then we would have received informa-

tion about the previous position. You may have been surprised that this was a function call. We didn't assign the return value of the function to a variable (i.e., we didn't say something equals the call to the function). We simply evaluated the function in an arithmetic expression. Functions can be used in assignments, comparisons, or arithmetic expressions. In other words, functions are like variables whose values are based on the parameters that are passed to them. You may then ask, "How do we know if we are in a long position or a short one?" Again, we refer to the EasyLanguage library and use the function marketPosition to determine our position. This function returns our current and previous positions based on the value that we pass it. If we had wanted our prior position, we would have passed a number one to the function: marketPosition(1). The marketPosition function will only return three different values: 1 for a long position, –1 for a short position, and 0 for a flat position. Now, let's go back to the code. Notice how the if-then structure alters the flow of the program. The liquidation orders are not issued unless we pass the test (logical condition). In order to pass the test in this case, three criteria must be met: (1) we have a position, (2) we have been in the trade for five or more days, and (3) the close of the day puts us into a losing position. Since we are using *and*, all of the conditional statements must evaluate to True before we can execute the line that immediately follows the if-then statement. If we had used logical *or*, then only one of the conditional statements would need to be true.

CONDITIONAL BRANCHING WITH IF-THEN-ELSE

The if-then statement provides only a single branch. Many times a program requires two branches: one branch that executes if True, the other if False. This type of conditional branching can be accomplished by using the *if-then-else* statement. If an evaluation of a conditional statement is True, the line or block of code that follows the *then* statement is executed. If the evaluation is False, the line or block of code that follows the else statement is executed. The syntax for the if-then-else is:

```
if(conditional statement) then [single statement]
else [single statement];

if(conditional statement) then
begin
       [multiple statements]
end;
else
begin
       [multiple statements]
end;
```

Notice that a semicolon is not used after a statement that precedes an else statement. If one is added, it will cause a syntax error. Let's incorporate our newly found knowledge into MyStrategy1. The following code causes our program to make a decision based on the number of days we have been in the trade. If we have been in the trade for less than five days, then we want to exit a long/short position on the lowest low/highest high of the past ten days. After we have been in the trade for five days or more, we will revert back to our previous exit strategy (come out of a losing position on the close after five days).

```
if (marketPosition = 1) then
begin
       if(barsSinceEntry(0)<5) then
       begin
              Sell("LongLoss") next bar at Lowest(low,10) stop;
       end
       else
       begin
              if(close<entryPrice) then Sell("LongLoss5Days") on this bar
              close;
       end;
end;
If (marketPosition = –1) then
begin
       if(barsSinceEntry(0)<5) then
       begin
              BuyToCover("ShortLoss") next bar at Highest(high,10) stop;
       end
       else
       begin
              if(close<entryPrice) then BuyToCover("ShortLoss5Days") on this
              bar close;
       end;
end;
```

Notice how in the last else block we didn't have enough room to put the entire statement on one line. Sometimes your statements will be too long to fit on one line of your screen. You can continue on as many lines as you need. The following statements are syntactically correct and should give you no problems during verification (compilation).

```
myValue1 = myValue2 +
           myValue3;

myValue1 = (close + high + low )/ 3 + Highest(high,numDaysBack) –
     Lowest(low,numDaysBack);
myValue1 =
     myValue2 +
```

```
          myValue3 +
               myValue4;
```

This is the beauty of having a line determination symbol (e.g., a semicolon). The compiler in EasyLanguage ignores all blank lines and spaces from the beginning of a statement to the semicolon. You can have 20 blank spaces or 20 blank lines. Our snippet of code not only exemplifies the if-then-else statement, but it also shows the concept of nested if-then statements. The if-then statements are separately located (nested) inside the scope (the boundaries of the statements that are controlled by the control structure) of the if(marketPosition = 1)-then statement:

```
If (marketPosition = 1) then
begin
       if(barsSinceEntry(0)<5) then    ← nested if-then
       begin
              Sell("LongLoss") next bar at Lowest(low,10) stop;
       end
       else
       begin
              if(close<entryPrice) then Sell("LongLoss5Days") on this bar
              close;
       end;
end;
```

With nested *if-then* statements, you can make your programs as complicated as you need. Now that we have an understanding of conditional branching, let's go back to the system that we introduced in Chapter 2 and see if we can understand what is going on. We will break up each module and explain what each statement is trying to accomplish:

```
{MyRsiSystem - trading strategy by George Pruitt 09/24/2001

Version 1.00   initial testing of my idea
Version 1.01   added RSI retracement component
Version 1.02   added profit objective mechanism
Version 1.03   changed the over bought/sold parameter

This trading signal is designed to buy when the RSI has double dipped into
the oversold territory and sells when the RSI has doubly risen into
overbought territory. Once a position is initiated an initial money
management stop and profit objective stop is invoked.}

Inputs: rsiLength(14), overSold(40), overBought(60), moneyManStop(1000);
Vars: myRsiVal(0),longProtStop(0),shortProtStop(99999),obCount(0),osCount(0);
Vars: longProfitStop(99999),shortProfitStop(0),takeProfitStop(2000);
```

We know that the program header is declaring and initializing the variables that we will need in the program. Since we are operating in one big loop, for each bar (daily or intraday) the following code is evaluated. First, we assign the myRsiVal variable the RSI value for this bar based on the closing price and a 14-bar period. This is simply a function call.

```
{Calculation Module}
myRsiVal = RSI(close,rsiLength);
if(myRsiVal > overBought and myRsiVal[1] < overBought) then
begin
     obCount = obCount + 1;
     osCount = 0; {Reset the OS counter since we are OB}

end;
```

After we get the current RSI value and store it in myRsiVal, we then test and compare it along with the previous bar's RSI value (remember how to do this?—we simply index the variable) against our overbought input parameter. Referring back to our program header, we see that this value is 60. If myRsiVal is greater than 60 and myRsiVal[1] (previous bar value) is less than 60, then we pass the test. Upon successfully passing the conditional statement, the program then flows into the block of code that immediately follows the then statement. Our variable *obCount* increases by one and *osCount* is reset to zero. Notice how we used self-descriptive names for these variables. Basically, we are counting the number of times we cross into overbought territory. Since we are dealing with the overbought region, we reset our oversold counter to zero. The next if-then statement counts the number of times we cross into oversold territory and resets our overbought counter to zero.

```
if(myRsiVal < overSold and myRsiVal[1] > overSold) then
begin
     osCount = osCount + 1;
     obCount = 0; {Reset the OB counter since we are OS}
end;
```

The following code pertains to our protective stop and profit target. If we are long, then we set our protective stop to the entryPrice – (moneyManStop/PointValue/PriceScale). This formula looks quite complicated, but it really isn't. We take our input parameter moneyManStop (currently set to 1000) and divide it by the instrument that we are testing's PointValue and then divide this result by the instrument PriceScale. Let's say we are trading coffee and we went long at 114.00. To calculate our protective stop, we would take 114.00 and subtract the result of ($1000/$3.75/100). This would give us 114.00 – 2.66 which equals 111.33. Many of you who are familiar with EasyLanguage and TradeStation know that there are built-in $ Stop Loss strategies that you can

add to any other strategy that will do the same as we have coded. The authors personally like to have all of the entries/exits in one strategy. Even if you don't use our programmed money management stops or profit targets, you will at least know how they are calculated. The keywords *PointValue* and *PriceScale* reflect the properties of the stock or commodity that you are currently testing. The keyword PointValue is the value per share of one increment of the PriceScale. The keyword PriceScale is the quoted scaling factor of the instrument. If you have two decimal places in the price, then your PriceScale would be 100. If you have four decimal places in the price, then your PriceScale would be 10,000. To figure out how a certain market is quoted (hundredths, thousandths, etc.), look at the price or divide 1 by the PriceScale.

```
if(marketPosition = 1) then
begin
       longProtStop = entryprice – (moneyManStop/PointValue/PriceScale);
       longProfitStop = entryprice + (takeProfitStop/PointValue/PriceScale);
       osCount = 0;   {Since we are long – reset the OS counter}
end;
if(marketPosition = –1) then
begin
       shortProtStop = entryprice + (moneyManStop/PointValue/PriceScale);
       shortProfitStop = entryprice – (takeProfitStop/PointValue/PriceScale);
       obCount = 0;    {Since we are short – reset the OB counter}
end;
```

We calculate our protective and profit target stops and then use them in our orders for the next bar.

```
{Order Placement Module}
if(obCount = 2) then sell short tomorrow at open; {We have entered OB twice}
if(osCount = 2) then buy tomorrow at open; {We have entered OS twice}

if(marketPosition = 1) then
begin
       Sell ("longLoss")next bar at longProtStop on a stop;
       Sell ("longProfit")next bar at longProfitStop on a limit;
end;
if(marketPosition = –1) then
begin
       BuyToCover ("shortLoss")next bar at shortProtStop on a stop;
       BuyToCover ("shortProfit")next bar at shortProfitStop on a limit;
end;
```

You may be asking, "Why did we put our liquidation orders inside if-then statements?" When you give TradeStation an order directive, it will execute automatically on the next bar. We don't know our protective stop or profit

target until we have a position. If we didn't use the if-then and we had a flat position and we issued a market order for the next bar, TradeStation would also issue a liquidation order for the next bar without knowing the correct stop or limit orders. Hypothetically, let's assume that obCount is equal to 2, and we place a market order to sell for the next day on the open. At the same time, we have also told TradeStation to cover a short position at either a protective stop or profit target limit. Sounds great, right? Wrong! We don't know our liquidation orders until after the market opens the next day. Without adding the contingency of being in a long or short position, we have put the cart in front of the horse. Remember, we are executing a single bar at a time and we don't know anything about tomorrow. So in our particular system the following will not work properly:

```
BuyToCover ("shortLoss")next bar at shortProtStop on a stop;
BuyToCover ("shortProfit")next bar at shortProfitStop on a limit;
```

This is a logical error, and it is the hardest and most difficult to correct. The compiler informs us of syntax errors and, in most cases, these types of errors are simple typos. Contingent orders must use some form of decision and, therefore, must be programmed with a program control structure.

REPETITIVE CONTROL STRUCTURES

Most trading strategies or indicators require a method of repeating a line or block of code. EasyLanguage usually automatically handles this necessity for the trader. However, there may be times when EasyLanguage doesn't have a solution and you will need to know about looping. A good example of looping can be found in the simple moving average calculation. In EasyLanguage, you can get the current moving average of a value by calling the Average function.

```
myAverage = Average(Close,10)   {This calculates a ten-day moving average}
```

See how EasyLanguage has provided a quick solution to our programming needs? Just for fun, let's see what is going on behind the scenes and learn about looping. EasyLanguage accomplishes iterative processing (looping) through two different iterative statements: the *for loop* and the *while loop*.

For Loop

Use the for loop when you know exactly how many repetitions are to be processed. In the case of the moving average calculation, and most other calculations, we know exactly how many bars we will use. Previously, we calcu-

lated a ten-bar moving average of the closing prices by calling the Average function. Now, let's do the same thing without a function call.

```
Vars: mySum(0),counter(0),myAverage(0);

mySum = 0.0;
for counter = 0 to 9 begin
       mySum = mySum + Close[counter];
end;
myAverage = mySum/10.0;

or

mySum = 0.0
for counter = 9 downto 0
       mySum = mySum + Close[counter];
end;
myAverage = mySum/10.0;
```

Remember EasyLanguage's ability to reference historical data by indexing the keywords *Open*, *High*, *Low*, and *Close*. We have incorporated that ability into our for loop. The historical closing information can be extracted by indexing the keyword *Close* with our counter variable (the name of this variable can be anything that is of numeric data type and can be used like any other variable). In the previous code, the program executes the block of code ten times:

```
for counter = 0 to 9 begin ← start at 0 and end at 9 inclusive – 10 repetitions
```

The first time the loop is executed, the counter is set to zero and EasyLanguage tests this value to the end value (in this case, nine). Since the counter is less than or equal to nine (it passes the test), the block of code immediately following the for statement is executed. Our variable *mySum* is initially set to mySum + Close[0] (the current closing price since the counter equals zero). When the program reaches the end of the block of code that is encapsulated by the for statement, it loops back up to the beginning of the for statement. EasyLanguage adds 1 to our counter (now counter equals 1) and compares it to the end value. This is similar in action to an if-then statement. Conditional branching is built-in to the for loop. Since we are still less than or equal to nine, we again execute the block of code that immediately follows the for statement. On the last go around, *mySum* was set to Close[0]. This time counter is now equal to 1, so *mySum* is now assigned the value of the previous mySum + Close[1]. Do you see where we are going with this? In programming lingo, *mySum* is known as an accumulator. The looping mechanism uses this variable to sum up the closing prices for the past ten days. The loop terminates after the counter reaches ten. You may think it would terminate when the counter

reaches nine, but the for-loop processes from the starting number through the end number. When the counter reaches nine, it is still less than or equal to the end value and, therefore, processes the loop once again. The next time through the loop, the counter is incremented to ten and then fails the test. Upon failure, control passes to the next line immediately following the for loop block. In our code, myAverage = mySum/10.0 is executed. After this statement is processed, we then have the average of the closing prices for the past ten days. In EasyLanguage, you can have for loops that loop from lower numbers to higher numbers or vice versa. If you loop from a higher number to a lower number, you must use the keyword *downto* or the program will not loop properly.

While Loop

The for loop works great when we know the number of repetitions that we want to process. There are times, however, when you don't know this information. Again EasyLanguage comes to the rescue with the *while loop*. The while loop continues to loop while a certain logical condition remains true. The following code illustrates the use of the while loop:

```
Vars: myTestDate(1010101),counter(0),myHigh(0),myLow(999999);
{This program calculates the highest price and lowest price since
the beginning of the year}
myHigh = 0;
myLow = 999999;
counter = 0;
while (Date[counter] > myTestDate)
begin
       if(High[counter] > myHigh) myHigh = High[counter];
       if(Low[counter] < myLow) myLow = Low[counter];
       counter = counter + 1;
end;
```

This loop starts with the current Date[0] and loops while Date[counter] is greater than the beginning of the year. EasyLanguage represents the date in a seven-digit format (YYYMMDD), whereas most other computer programs represent the date in an eight (YYYYMMDD)- or six (YYMMDD)-digit format. If the today's date were 20010101, EasyLanguage would store it as 1010101.

Calendar Date	EasyLanguage Representation
19991231	991231
20000101	1000101
20010101	1010101

By increasing the counter by 1 each time through the loop, we are actually going back in history. Remember, today's open price is Open[0] and yesterday's open is Open[1]. As we loop backward, we compare the highs and lows of the bars with the *myHigh* and *myLow* variables. Each time we find a higher high or lower low, we store those prices in our variables for later use. The while loop is not terminated until we go back in time to the date of 20010101. We were able to loop without knowing the exact number of repetitions that needed to be processed.

CONCLUSIONS

Through the use of program control structures, a program can make an informed decision. Remember, the programmer makes the computer intelligent by setting up the decision process and the consequences of the decision. Decision-making is programmed through the use of *if-then* and *if-then-else* statements. These statements control which instructions are executed and which are not. The *for loop* and *while loop* statements are two program control structures that provide us with one of the great benefits of computers: the ability to do repetitive tasks quickly. Understanding how computer programs make decisions is vital in our goals of accurately programming our analysis techniques.

4

TradeStation Analysis Techniques

EasyLanguage is the power behind TradeStation. Without this power, TradeStation would not be any different than any of the other real-time charting software packages. EasyLanguage is the driving force behind all of TradeStation's analysis techniques: PaintBar, Indicator, ShowMe, Function and Strategy. Each of these techniques are discussed, and sample code is given so that you can use it as a template for your own research. We have covered the basic essentials of programming from the first three chapters and now will use this knowledge to put our trading ideas into action.

INDICATORS

An *indicator* is a graphic representation of a mathematical formula used to analyze market data. If you have used charting software before, you probably know what some of the most popular indicators are. Indicators are the lines (graphs) that are either plotted on top of bar charts or in subgraphs above or below the bar chart. These lines are used to help understand and, in some cases, to forecast market action. Indicators are the plotted output of a formula applied to the price data.

The most famous indicator of all is the moving average of closing prices. The graph of the moving average is a continuous line that can be plotted on the actual bar chart data. Not all indicators can be plotted on the underlying bar chart due to the difference in their scaling properties. The scale of the output of a moving average calculation is the same as the data that was used for the input. Since the scale of the input and output of this indicator is the same, it

can be plotted on the same one as the price data. The outputs of some indicators, such as the relative strength index (RSI, by Welles Wilder), have a different scale than price data and should be plotted in a subgraph. However, with TradeStation you can graph these types of indicators on the bar chart if you like. Keep in mind that the graphs have different scales. The output of most indicator functions does not create smooth-looking, continuous graphs. They generate a data point for each bar of data and then TradeStation connects these lines together in a connect-the-dots fashion.

To demonstrate how simple it is to apply an indicator to a bar chart, let's go ahead and apply the Mov Avg 1 Line indicator to a Yahoo! daily bar chart. Create a daily bar chart of 500 days for Yahoo! and then go up under the Insert menu and select Indicator. A dialog box like the one in Figure 4.1 should open.

Scroll through the list of indicators and select Mov Avg 1 Line. Make sure the Prompt for Format box is checked and then click OK. Since we checked the Prompt for Format box, you will be presented with another dialog box that should be like the one in Figure 4.2.

This dialog box allows the user to change the format of the Mov Avg 1 Line indicator. Here the width, color, and style of the line and the inputs (if any) of the indicator can be changed. Click on the Scaling tab and you will have a choice of four different scaling types:

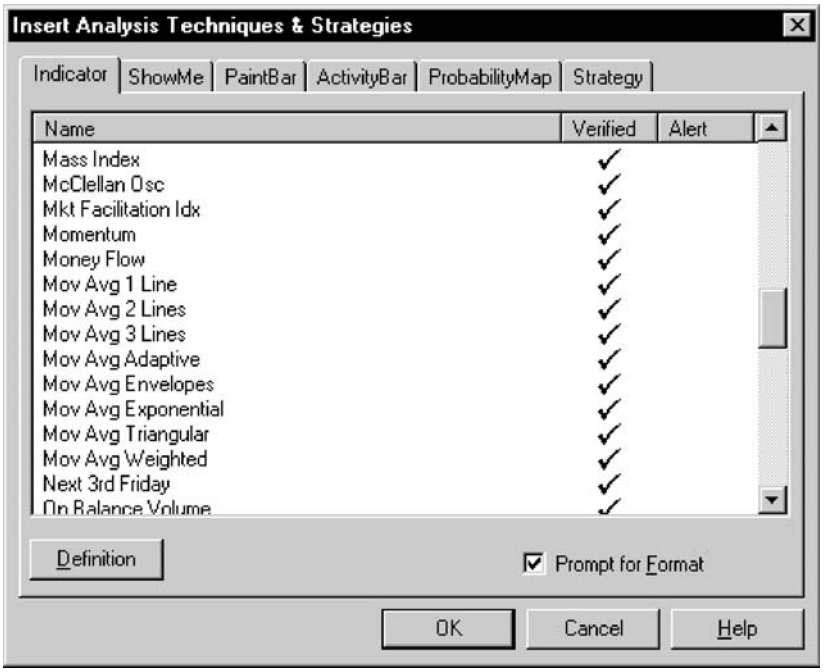

Figure 4.1 Insert Analysis Techniques and Strategies—Indicator

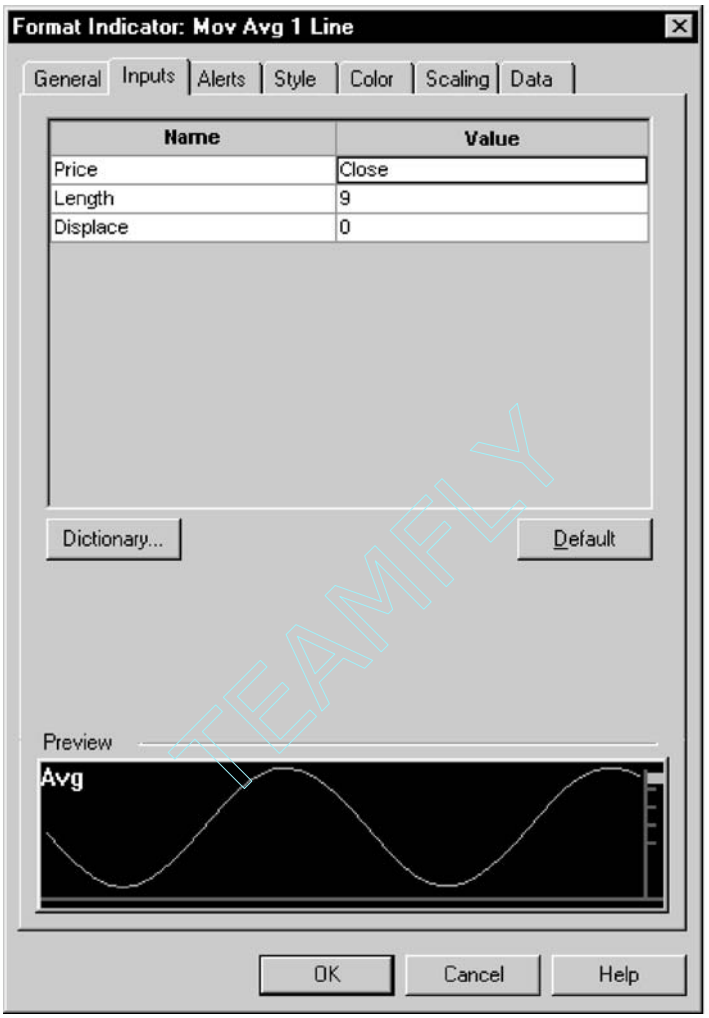

Figure 4.2 Format Indicator: Mov Avg 1 Line

Same as symbol: Choose this option to use identical scaling for the y-axis (price axis) and the symbol to which the analysis technique is applied.

Screen: Select this option to display only the range of values currently displayed on the screen without regard to the range of data loaded for the chart.

Entire data series: Choose this option to display the entire range of values for all data loaded in the chart, no matter how far you scroll to the left or right of the current screen view.

User-defined: Select this option to format the y-axis to your preferences. Enter the lowest value to use in the Minimum box and the highest value in the Maximum box.

We prefer to use "Same as symbol" scaling when the indicator is to be plotted on top of the bar chart. If an indicator is oscillator-based and is plotted in a subgraph, we generally use "Same as symbol" or "Screen" scaling. Check the Same as symbol box and then click OK. You should have a chart window similar to Figure 4.3 on your screen.

You can scroll through the data and the indicator will change based on the data that is currently in your view. You may want to play with changing the format of this indicator or inserting more of the built-in indicators as practice.

Applying an indicator requires a few mouse clicks and pecks at the keyboard. Programming an indicator can be as simple if you understand the basic framework and the keywords or functions that are used in the development of this type of analysis technique. Let's create our first indicator. Make sure TradeStation is up and running and go under the File menu and select New. A dialog box like the one in Figure 4.4 will open.

You may remember this dialog box from Chapter 1. Select the EasyLanguage tab and select the Indicator icon and then click on the OK button. The *New Indicator* dialog box will open and ask you to type in the name of the indicator and some notes about the indicator. Type "MyMovAvgIndic" in the name field and "simple moving average indicator" in the notes field. This dialog box will also ask you to make the analysis technique available to a chart analysis. We *always* check this box. Finally, you can choose what type of template from which to use when building your indicator. Select none and then click OK. Once you become an EasyLanguage pro, you will probably want to use a programming template. These templates provide some code to get you

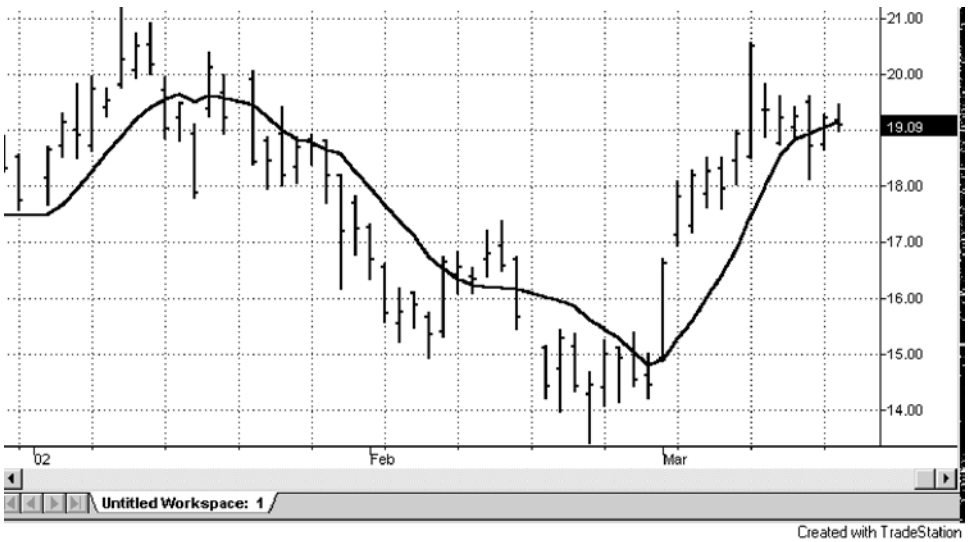

Figure 4.3 Yahoo! Bar Chart with Moving Average Indicator

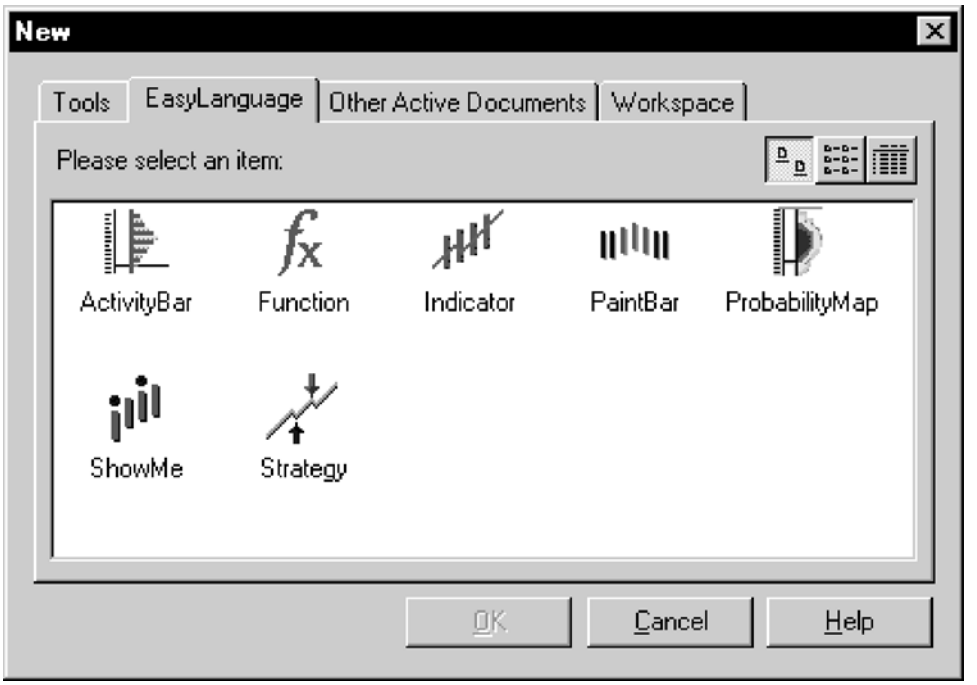

Figure 4.4 New Dialog—Indicator

up and running quickly. You should be presented with a blank window titled *Indicator: MyMovAvgIndic*. TradeStation has initially created an indicator based on our input from the previous dialog boxes. Type the following code exactly as you see here:

```
Inputs: myPrice(close),myLength(9);
Vars: counter(0),sum(0.0),myAverage(0.0);
sum = 0.0;
for counter = 0 to myLength-1
begin
       sum = sum + myPrice[counter];
end;
myAverage = sum/myLength;
Plot1(myAverage, "SimpAvg1");
```

After you are finished typing, hit the F3 key and verify your code. Again, if you get any error, double-check your code for any typos. Once you are successful at verifying your code, go ahead and apply MyMovAvgIndic to the same Yahoo! daily bar chart that we used previously. Click OK in the associated dialog boxes until you have the moving average line overlaying the price data. It should look similar to the chart in Figure 4.5.

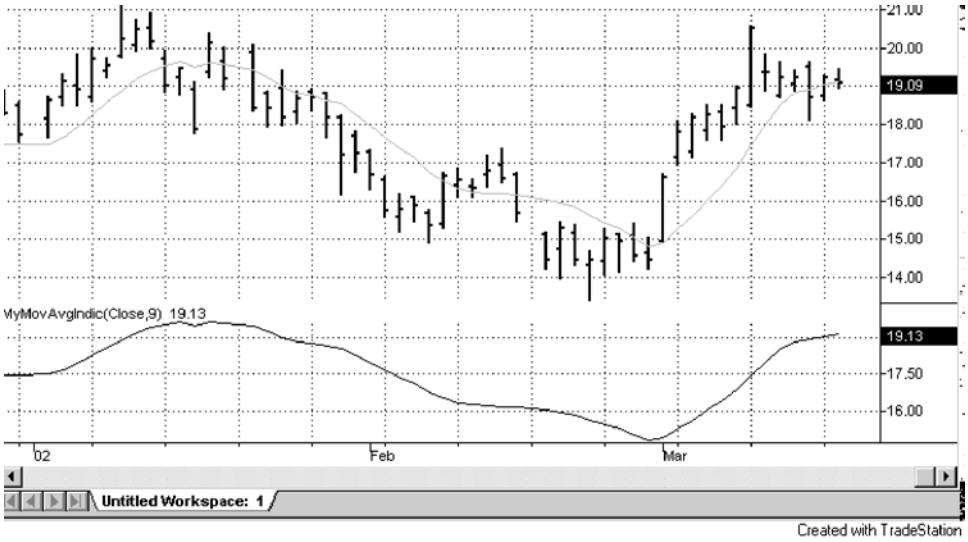

Figure 4.5 Yahoo! Bar Chart with MyMovAvgIndic

The key to a well-written indicator is its flexibility. In MyMovAvgIndic we utilized two inputs: myPrice and myLength. These two inputs allow the user the freedom to pick what data (Open, High, Low, Close, or Volume) and how many days to use in the calculation. We have designed this indicator to be about as flexible as possible. We could have programmed the indicator with only the myLength input and forced the user to always use a hard-coded (programming designed to get a particular job done without regard to future flexibility) price series. If we hadn't allowed the user to pick which data series to use, we would have had to program the indicator in the following manner:

```
for counter = 0 to myLength-1
begin
       sum = sum + Close[counter];
end;
```

Instead of using myPrice in the for loop, we would have to hard code the Close or High or Low or Open price series into our programming. Our indicator would not be universal, and you would have to have four different indicators for each price series. The previous programming allows MyMovAvgIndic to be universal. We can calculate the moving average of the Highs, Lows, Opens, or Closes of any market by simply changing our first input, myPrice.

In the discussion of indicator flexibility, we brought up the term *series*. Remember in Chapter 1 when we introduced data types? There were basically three different types: numeric, Boolean, and string. The numeric type can be

divided into two different categories: *simple* and *series*. A variable of type numeric simple has only one value at a time. In the code of MyMovAvgIndic, the variable sum is of this type. A variable of type numeric series is one that has a list of different values. The keywords *Open*, *High*, *Low*, and *Close* are of this type. We can index these variables and get different values (e.g., Close[1] is yesterday's closing price and Close[2] is the previous day's closing price). In computer lingo, these variables are known as *arrays*. An array is a list of items that have something in common. Like most other high-level programming languages, EasyLanguage allows you to define variables as arrays. The average trader seldom uses arrays beyond the built-in array variables, so we will temporarily put this discussion on hold. Just know that when we say a price series, we mean a list of the entire history of that particular price (Open, High, Low, and Close).

Let's review the rest of the code. We know the inputs and their functions and for loops (Chapter 3) and how to calculate the moving average of prices. The line of code that makes this program an indicator is:

```
Plot1(myAverage, "SimpAvg1");
```

This statement tells TradeStation to plot the value of the variable myAverage in a chart window below the chart of the price data. Once the indicator is plotted, we simply drag it from the subgraph onto the same graph as the price data and use the same scaling. You can see where price crosses the moving average line. To determine the exact date that a price bar crosses from above or below the moving average line, you must set the scale of the two graphs to be exactly the same.

The *Plot1* keyword acts in similar fashion to a function call. We passed it a list of parameters and it did something for us. It can also return the value of the plot from the most recent bar that was executed. Plot1 is different than the functions that we have discussed in that you can pass it a different number of parameters. In our example, we simply passed the value that we wanted to plot and the name of the indicator. This name shows up in the information window that pops up when you click on a bar in a chart. We could have passed more or less parameters to the Plot1 statement:

```
Plot1(Value) ;
```

or

```
Plot1(Value, "My Plot Name", Red, Default, 0) ;
```

Plot1 can take up to five parameters: (1) the value to be plotted, (2) the name of the plot (optional), (3) the color of the plot (optional), (4) background color (optional), and (5) the thickness of the line that represents the plot. EasyLan-

guage only requires the value to be plotted in the Plot1 statement; the other arguments are optional. If you don't pass a particular argument, EasyLanguage presumes you want to use the default value. Even though some of the arguments are optional, the order of the arguments is important.

EasyLanguage expects the first argument to be the value to be plotted, and the second argument to be the name of the plot and so on. This order has to be clear. Let's say you wanted to pass the Plot1 statement the value to be plotted, its name, and the width of the line. You would have to pass all of the arguments because the width argument is last in order. You would have to invoke the Plot1 statement by typing:

```
Plot1(myValue,"myValueName",Default,Default,5);
```

The keyword *Default* tells TradeStation to accept whatever default values exist for that particular argument. In this case, they are the color of the plot and the background color. EasyLanguage is smart, but it is not a mind reader. It knows which values are being used by their place in the argument list.

Indicators are powerful tools in the analysis of data and the design of trading strategies. If you want to include an indicator in a strategy, you can quickly plot the indicator and visually evaluate its effectiveness. Unfortunately, as we explained in *The Ultimate Trading Guide* (Hill, Pruitt, and Hill, John Wiley, 2000), most canned (included for free) indicators are not effective by themselves.

PAINTBAR AND SHOWME STUDIES

These two studies are similar in that they both look for and mark a bar or bars that meet specific criteria. The PaintBar study will mark the entire bar, whereas the ShowMe study will usually place a mark above or below the bar. These studies make it easy to visually interpret different market conditions; for example, you can paint bars one color when the market is overbought and another color when it is oversold. ShowMe studies are best used for criteria that result in a low frequency of occurrences. The less dots that are above or below a bar, the easier it is to visually interpret the market activity following the occurrence of the certain criteria. Pivot highs and pivot lows are best illustrated with a ShowMe study rather than a PaintBar study. The choice of which study to use is up to the user's own preference. We prefer to use ShowMe studies for pattern recognition and PaintBar studies for illustration of certain market modes. The best way to learn to program these analysis techniques is to jump in and create them from scratch. Let's start with a PaintBar study.

With TradeStation running, go under the File menu and select New. The now familiar dialog box will come up. Select the EasyLanguage tab and

click on the PaintBar icon. In the resulting dialog box, type "MyPaintBar" in the name field and "paint bars different colors in OB/OS regions" in the notes field and then click OK. You will be presented with a blank, yet familiar EasyLanguage window. Type the following code in exactly.

```
{Simple PaintBar Analysis
Demonstrates TradeStation's ability to highlight bars in
different market conditions}

Inputs: overbought(70),oversold(30),rsiLength(14);
Vars: myRsiVal(0);

MyRsiVal = RSI(Close,rsiLength);

if(myRsiVal>overBought) then PlotPaintBar(High,Low,"RSI",RED);
if(myRsiVal<overSold) then PlotPaintBar(High,Low,"RSI",YELLOW);
```

After typing the code, hit the F3 key or go up under the File menu and select Verify. Let's see how we did. If you don't have the chart of Yahoo! that we used earlier in the chapter open, go ahead and create another one. Insert our PaintBar study in the same fashion as we did our MyMovAvgIndic (under the Insert menu). (You will notice the further you get into this book, the less descriptive we become in our instructions. We feel this promotes your ability to remember the processes necessary to use TradeStation.) Once you have inserted MyPaintBar into the chart, you should be able to scroll through the chart and see the yellow and red daily bars. If you would like to see if our PaintBar study is accurate, go ahead and insert the RSI indicator with our same inputs. We should have red bars when the oscillator is in the overbought region and yellow bars when it is in the oversold region.

The code for our PaintBar study is similar in structure to our moving average indicator that we programmed earlier. The only thing that differentiates a PaintBar from an Indicator is the PlotPaintBar statement.

```
PlotPaintBar(High,Low,"RSI",RED);
```

This statement tells TradeStation to paint a bar red from the high price to the low price. Also, the statement instructs TradeStation to use the word "RSI" in the bar information window. If you hold the mouse button down and scroll across a bar that has been painted, you will get a RSI1 and RSI2 value in the bar information window. These values represent the high and the low prices of the bar. We used the word *RSI* to denote that we were using the RSI indicator to determine our overbought/oversold conditions. You can use any word you like. We painted our overbought bars red by telling TradeStation to use the color red in our PlotPaintBar statement. This argument is optional; if you don't choose a color, TradeStation will choose for you. We wanted to paint

bars two different colors based on market conditions, so we chose the colors ourselves. When working with analysis techniques, you can specify any of the 17 colors listed below (including -1), using the name, or numeric equivalent:

Color Name	Numeric Equivalent
Default	–1
Black	1
Blue	2
Cyan	3
Green	4
Magenta	5
Red	6
Yellow	7
White	8
DarkBlue	9
DarkCyan	10
DarkGreen	11
DarkMagenta	12
DarkRed	13
DarkBrown	14
DarkGray	15
LightGray	16

In addition to selecting the PaintBar color, you can also select the portion of the bar that you want painted. In our PaintBar study, we painted the entire bar from top to bottom. We could have painted the bar from the Open to the High, Open to Close, Open to Low, or any combination of these. You can't paint from Open to Open, High to High, Low to Low, or Close to Close; there must be a range. Using the PlotPaintBar statement is quite easy. The hard part is creating the criteria necessary to determine when and when not to use the statement. When you typed the code for MyPaintBar, we used an if-then construct to alter the flow of the program. We only painted a bar red when the RSI was above 70. We only painted a bar yellow when the RSI was below 30. We can quickly scan this chart and determine potential overbought and oversold market conditions. If you don't care for the RSI as an indicator for overbought/oversold, you could incorporate a different indicator.

You may ask, "Why use the PaintBar study when you could have just simply used the Indicator?" We programmed our study utilizing the RSI function for demonstration purposes. Many times a PaintBar study doesn't use some

built-in function or indicator. We have programmed a complex pattern recognition PaintBar to illustrate this point. The coding for this PaintBar will draw on everything we have learned up to this point. Go ahead and create a new PaintBar and name it MySequentialHunter. In the notes field type, "paint sequential set up". This study is based on one of Tom DeMark's complex patterns, the Sequential. We will program the first stage of the pattern and paint the bars that fall within our pattern recognition criteria. Type the following into your EasyLanguage document and verify it.

```
{Paint the Sequential Setup Pattern
Sequential is by Tom Demark
Paint buy setup Yellow and sell setup Red}
Vars: index(0),checksum(0);

{Buy Setup}
checkSum = 0;
for index = 0 to 8
begin
       if(Close[index] < Close[index+4]) then checkSum = checkSum + 1;
end;
if(checksum = 9) then
begin
       for index = 0 to 8
       begin
             PlotPaintBar[index](High[index],Low[index],"Sequential",YELLOW);
       end;
end;

{SellSetup}
checkSum = 0;
for index = 0 to 8
begin
       if(Close[index] > Close[index+4]) then checkSum = checkSum + 1;
end;
if(checksum = 9) then
begin
       for index = 0 to 8
       begin
              PlotPaintBar[index](High[index],Low[index],"Sequential",RED);
       end;
end;
```

After you successfully verify your code, apply MySequentialHunter to the same YAHOO! chart that we have been using. You will probably want to delete the other analysis techniques before inserting this one. The code for MySequentialHunter may look daunting, but it is relatively simple. We have four for

loops. The first loop counts the consecutive days that the closing price is less than the closing price four days prior. If we have nine consecutive closes that meet our criteria, then the *checkSum* variable will be equal to nine. If we don't have nine consecutive closes, then *checkSum* will be less than nine. If *checkSum* is equal to nine, we then proceed to the next for loop. This loop calls the PlotPaintBar statement nine times (our index variable goes from 0 to 8). The first time through the loop, the index is equal to zero. Therefore, the call to the PlotPaintBar looks like:

```
PlotPaintBar[0](High[0],Low[0],"Sequential",YELLOW);
```

The next time through the loop, the call looks like:

```
PlotPaintBar[1](High[1],Low[1],"Sequential",YELLOW);
```

This continues until the termination of the loop. By using the brackets and an index variable, we are telling TradeStation to paint today's bar yellow, yesterday's bar yellow, in addition to the preceding seven bars yellow. Again, if you want to paint yesterday's (or prior) bar yellow you must index the PlotPaintBar statement with [1] (PlotPaintBar[1]—basically we are offsetting the bar that we want painted from the current bar). You also must be careful to synchronize the High's and Low's index variables with the same PlotPaintBar index variable. The following statement may not produce your expected outcome:

```
PlotPaintBar[1](High,Low,"Sequential",YELLOW);
```

This tells TradeStation to paint today's high and low values on yesterday's bar. This may or may not paint the bar the way you want it to be. Just remember to paint the correct high and low price for the corresponding bar in the past.

Most analysis techniques are symmetric; a bullish analysis is usually just the opposite of a bearish analysis. Referring back to the code for MySequentialHunter, you will notice that there are only two lines that separate the buy setup from the sell setup:

```
Buy - If(Close[index] < Close[index+4] then checkSum = checkSum + 1;
Sell - If(Close[index] > Close[index+4] then checkSum = checkSum + 1;
```

The only other difference is the call to the PlotPaintBar statement with different colors. Once you program the bullish analysis technique, you can quickly create the bearish analysis by simply copying, pasting, and editing the logic. Many times you can accomplish this by simply changing Highs to Lows, greater than to less than, positive numbers to negative, and so on. Of course, this wouldn't work for an analysis that wasn't symmetric.

The ShowMe study is similar to the PaintBar. We won't bore you much further with an elaborate explanation, so let's learn by doing. Create a new ShowMe study in similar fashion to a PaintBar study and type "MyShowMe" in the name field, and "Show a two-day flip" in the notes field. Once you get a blank EasyLanguage document, type the following in exactly.

```
{MyShowMe
Show the Bars that represent a
Two-day flip}

Inputs: strongThrustPrcnt(.75),volCalcDays(10);
Vars: volMeasure(0);
volMeasure = avgTrueRange(volCalcDays);

{bearish indicator}
if(Close[1] > Open[1] + strongThrustPrcnt*volMeasure and
   Close < Open – strongThrustPrcnt*volMeasure and
   Open < Close[1]) then
   begin
       Plot1(High,"TwoDayFlip",RED);
       Plot1[1](High[1],"TwoDayFlip",RED);
end;

{bullish indicator}
if(Close[1] < Open[1] – strongThrustPrcnt*volMeasure and
   Close > Open + strongThrustPrcnt*volMeasure and
   Open > Close[1]) then
   begin
       Plot1(Low,TwoDayFlip",YELLOW");
       Plot1[1](Low[1],"TwoDayFlip",YELLOW");
end;
```

The programming of a ShowMe study is almost the same as a PaintBar study—the only difference is the ShowMe study uses Plot1 and the PaintBar uses the PlotPaintBar statement to draw to the screen. In our ShowMe study, we mark the two bars that make up the Two-Day flip pattern. As we did in our PaintBar, we programmed our setup criteria by using if-then program control structures. We only mark the bars that pass our test. The Two-Day flip can indicate an end to a bullish or bearish trend. The bullish Two-Day flip occurs when yesterday's close is well below yesterday's open, today's open is below yesterday's close, and today's close is well above today's open. The bearish Two-Day flip is just the opposite of the bullish version: yesterday's close is well above yesterday's open, today's open is above yesterday's close, and today's close is well below today's open. See the symmetry? This is a general description of the pattern. Programmers like us (if you have made it this far and understand the concepts that we have presented—then classify yourself as a programmer)

know that a general description of an idea is not sufficient to produce actual code. Therefore, the exact definition of a bullish Two-Day flip is: yesterday's close is less than one 10-day average true range below yesterday's open, today's open is less than yesterday's close, and today's close is greater than one 10-day average true range above today's open. Again, the bearish version is just the opposite. This exact mechanical description is necessary for successful translation into EasyLanguage. The Plot1 statement that we use in our ShowMe study is the exact same statement that we used in our Indicator. Instead of creating a continuous line that represents an output of a formula, we are using the statement to demark a bar that meets a certain criterion. In our ShowMe code, we used the Plot1 statement to mark the high price or the low price of the two bars that made up the Two-Day flip:

```
Plot1(High,"TwoDayFlip");
Plot1[1](High[1],"TwoDayFlip");
```

As with the PaintBar study, we had to synchronize our Plot1 statement with the corresponding bar. When plotting an Indicator, you use the value of the indicator and not a reference to a bar's specific price. Once we encounter a day that does not pass our criteria, the drawing stops. Hence, you have dots instead of a continuous line. You can have up to four different Plot statements in your code: Plot1, Plot2, Plot3, and Plot4. Each plot statement can represent a different value; Plot1 could be a moving average value, whereas Plot2 could be a Bollinger Band. The key to an accurate PaintBar or ShowMe study is found in the programming of the criteria.

FUNCTIONS

Functions are the backbone of TradeStation. One of the most important concepts to efficient programming is the idea of reusable code. How would you like to have to reprogram Welles Wilders' RSI calculation every time you incorporated it into your analysis technique? Programming analysis techniques require a heavy dependence on EasyLanguage's vast library of functions. Many of the indicators and strategies that you will develop will mostly consist of function calls. You can think of the functions that make up the library as small building blocks and yourself as a mason. It is your job to build your analysis techniques out of these blocks. Of course, there will be times when a mason must customize a block to fit a particular project. This applies to programmers also; you might find that by tweaking the built-in Stochastic function you end up with a better oversold/overbought indicator. By tweaking, we mean changing a portion of the code that makes up the function. If you ever do this, always remember to save your modified function under a different name than the

original. If you change the RSI calculation, rename it to something like MyRSI. You want to keep the built-in library as pristine as possible. Many times you will borrow or purchase code from other TradeStation users, and in most cases this code will presume that the built-in library is the same as the one from the computer where it was created. If your library is different than the creator of the code's library, you may get totally different results.

As a programmer, you are not limited to using functions or simply editing existing ones. You can create these building blocks from scratch. The more you program, the more intricate your programming becomes. Elaborate programs require elaborate building blocks. In the early stages of learning how to program analysis techniques, the built-in library is more than sufficient. As you progress, you will soon discover that the library is too limiting and must be expanded. You expand the library by creating functions. Since, right now, you may not be an experienced programmer, you may be asking yourself, "How will I know when I need to create a function?" There are basically two situations when you will need to be a function creator. The first is when you discover that you are repetitiously typing the same code over and over again. The second is when you discover the Holy Grail and you would like to pass this information on to other users for free. For free? We mean $10,000 a pop.

As we have done with other analysis techniques, let's create a function from scratch. Go up under the File menu and go to New. When the dialog box opens, select function. Type "ChoppyMarketIndex" in the name field, and "This function determines market choppiness" in the Short Description field. As you can probably deduce from the name of this function, we are attempting to measure a market's indecisiveness. You will notice that we must inform TradeStation of the type of data that our function will return. All functions must return a value (the result of the function's calculations). Most of the time, a function will either return a numeric or Boolean value. A function can also return a string value (a letter or a string of letters), but we have never encountered the need for this. Click on Numeric simple and hit OK. When you have a blank EasyLanguage document, type the following code:

```
{Choppy Market Index Function
This function returns a value from 0 to 100.
A lower value denotes higher market indecisiveness (choppiness),
whereas a higher value denotes a trending market.
The only input is the number of bars that we look back.}

Inputs: periodLength(Numeric);
Vars: num(0),denom(1);
if(periodLength<>0) then
begin
       denom = Highest(High,periodLength) – Lowest(Low,periodLength);
       num = Close[periodLength-1] – Close;
```

```
        num = AbsValue(num);
        ChoppyMarketIndex = 0.0;
        if(denom<>0) then ChoppyMarketIndex = num/demon*100;
end;
```

Did you notice how this function was made up of other functions (building blocks)? We calculated the denom (denominator) by using the Highest and Lowest functions. We calculated the num (numerator) by using AbsValue (returns the absolute value of a number) functions. The only confusing snippet of code in this function is probably:

```
Close[periodLength-1] – Close
```

You may be asking why we subtracted 1 from the periodLength. This is a great question. If you incorporate today's closing price into a calculation, then the closing price 30 days ago would be referenced by Close[29]. Remember that Close[1] is yesterday's closing price not today's. Since we want our index to flow between 0 and positive 100, we remove the negative sign of a down move in the market. We are only interested in absolute distances.

Becoming a good EasyLanguage programmer does not require an in-depth knowledge of the intricacies of all of the built-in functions. It does, however, require knowledge of how to put all of the pieces together. (Maybe we really should call ourselves masons instead of programmers.) The function that we just typed in and verified can be used to determine the current choppiness of a market. Later on when we begin to develop successful trading strategies, we will use this function as a building block. The ChoppyMarketIndex function determines market choppiness by dividing the change in market price for the past 30 days by the total distance the market has traveled during that time period. If the net change in market price is small and the market has demonstrated wild swings, the ChoppyMarketIndex function will return a small number. In the following description of our function, we assume the periodLength is equal to 30. We determine the price change for the past 30 days by subtracting the closing prices of 30 days ago from today's closing price. The variable num is assigned this value. We then determine the total distance the market has traveled by subtracting the lowest low for the past 30 days from the highest high of the past 30 days. The variable denom is assigned this value. The ChoppyMarketIndex is then assigned the value of num divided by denom multiplied by 100. Since the change in market price for the past 30 days will always be equal to or smaller than the total distance the market has moved over the same time period, our function will always return a number between 0 and 100. You will notice that we have made the function flexible. We used an input as the period length. By doing this, we have allowed the user of this function the ability to change the number of days that is used in the calculation. If you are developing a short-term strategy, then you would probably pass the

function a short period length and vice versa. Once you have finished your calculations for a function, you must assign a value to the name of the function. In this case, we assigned our final calculation to ChoppyMarketIndex, the name of our function. If you forget to do this, the function will not return an accurate value. This function is actually useful and will be the basis for one of the strategies that we develop in Chapter 6. Did you notice that in the Inputs statement of this function we used *NumericSimple* inside the parentheses of our Input variable, periodLength? "Why did we do this and why didn't we simply type a default value for the input variable?" you may inquire. Again, this is a wonderful question and since functions are the backbone of TradeStation, it should be answered in great detail.

Functions are in some ways similar to other Analysis Techniques, but overall they are quite different. Think of functions as separate subprograms. These subprograms are designed to calculate and change the value of some variable. Indicators, ShowMe, and PaintBars are designed to graphically represent an idea or a mathematical expression. When we create a function, TradeStation needs to know ahead of time what type of variable the function would return. TradeStation also needs to know the type of information that is being passed to the function, which is why we used the keyword *NumericSimple* in the Input statement. The function's Input statement is its interface with the program or analysis technique that called it. It is the doorway that data and information passes. In computer lingo, it is also known as the formal parameter or argument list. Like the function itself, a parameter can be of subtype simple, series or reference. We haven't discussed the type reference, but we will due to its powerful capabilities.

Simple parameters are constant values that are set in the trading strategy or analysis technique that calls the function. Our periodLength parameter is of this type. Simple parameters require less memory and improve overall execution speed. They retain their values within the function and cannot be modified within the body of the function. In other words, you can't change their value once inside the body of the function, period.

Like simple parameters, series parameters are constant values that are set in the trading strategy or analysis technique that calls the function. However, when the function refers to previous values of the input variable you use as the parameter, then this parameter must be defined as a series parameter. Current and historical values of the input variable are accessible from within the body of the function. This allows the function to refer to the previous bar's value of the parameter. Confusing, isn't it? Let's say you create a function that sums the differences between two data or price series. To make the function as flexible as possible, you want to give the user or caller of the function the ability to choose which data series is used. With this in mind, you would program the function to accept NumericSeries type data. Take a look at the following code as it might help clear up some of this confusion.

```
{SumDiff – sum up the difference between two data series}
Inputs: dataSeries1(NumericSeries),dataSeries2(NumericSeries),length
     (NumericSimple);
Vars: sum(0);
for Value1 = 0 to 20
begin
       sum = sum + (dataSeries1[value1] – dataSeries2[value1]);
end;
SummDiff = sum;
```

See how we used the keyword *NumericSeries* in the function's Input statement. This prepares TradeStation, ahead of time, to accept the history of the variable that is passed to the function. In computer lingo, this preparation is known as function prototyping. So, to summarize, a simple parameter is just one value, whereas a series parameter is a list of values. Like simple parameters, series parameters cannot be changed within the body of the function.

Parameters can be passed to a function by value or by reference. When the parameter passes information by value, as is the case with simple and series type parameters, the function creates a copy of the information passed into it, and whatever is done with the parameter in the function does not affect the value of the parameter within the trading strategy or analysis technique that called the function. When information is passed by reference, the function uses the original information from the trading strategy or analysis technique that called the function, and any calculations that are performed on the parameter are reflected in the value of the parameter within the analysis technique that called the function as well as within the function. Why on Earth would you want to do this? Remember how we are always comparing EasyLanguage to modern-day, high-level programming languages? Most programming languages have two types of subprograms: functions and subroutines. Functions are used when a calculation returns only one value. Subroutines are used when more than one value is calculated or a chore is needed to do more than just simply return a value of a calculation. The programming that went into the creation of TradeStation uses functions and subroutines. When you click on a menu, a subroutine is called to handle the actual drawing of the menu list and another subroutine is called to handle whatever command you choose from the menu. These subroutines do more than just return single values. Since we are not programming a graphical user interface, we will simply use the reference type and functions to return multiple values. In all honesty, you probably will only use reference-type parameters on rare occasions. The following function demonstrates the proper use of these types of parameters.

```
{Function ZoneBands by George Pruitt –
       Illustrates the power of passing parameters by reference.}
Inputs: zBand1(NumericRef),zBand2(NumericRef),
       zBand3(NumericRef),zBand4(NumericRef),length(NumericSimple);
```

```
Vars: myAverage(0),myAtr(0);
myAverage = Average(Close,length);
myAtr = AvgTrueRange(length);
zBand1 = myAverage + (myAtr/2);
zBand2 = myAverage + myAtr;
zBand3 = myAverage – (myAtr/2);
zBand4 = myAverage – myAtr;
ZoneBands = 1;
```

This function does not seem to return a significant value, however, it does return a value out of pure necessity. All functions must return a value. In the case of ZoneBands, the statement that assigns a value to the function name is only there for syntactical correctness. This function in fact returns four different and significant values. Maybe we shouldn't use the term return. This function modifies the four parameters (zBand1, zBand2, zBand3, zBand4) that are passed as NumericRef type. Once they are changed within the body of the function, they are forever changed. We were able to calculate four variables from one function call. The following snippet of code illustrates how to use this type of function call.

```
Vars: myBand1(0),myBand2(0),myBand3(0),myBand4(0),tempReturnVal(0);
TempReturnVal = ZoneBands(myBand1,myBand2,myBand3,myBand4,20);
Plot1(myBand1,"Band 1");
Plot2(myBand2,"Band 2");
Plot3(myBand3,"Band 3");
Plot4(myBand4,"Band 4");
```

Every time ZoneBands is called, the four variables are modified. We assigned a temporary variable the value that is returned from the call to ZoneBands.

STRATEGIES

Strategies are a major part of this book. Indicators are used more or less as tools, whereas strategies are used as the mechanisms to generate exact buy and sell signals. Strategies are the vehicles that most third-party developers use to sell their ideas. You may not believe us when we say that a good portion of TradeStation users only use their software as an interface for these strategies for hire. We don't have a problem with this. TradeStation can be used as a research tool or as a tool for information dissemination.

As you have seen from the previous chapters, strategies are programmed in similar fashion to other analysis techniques. They all use data types, incorporate assignments and/or decisions, and do mathematical calculations. The one thing that separates strategies from other analysis techniques is they issue orders. As we have discussed, TradeStation can place three types of orders:

market, limit, and stop. Market orders can be placed *this bar on close* or *next bar at market*. Strategies can place multiple orders for any single bar. TradeStation has two rules that determine which orders get filled:

1. Orders on Close and Next Bar at Market
2. Stop and Limit Orders

Orders placed to be filled *this bar on close* have the highest priority. Once all of these orders have been filled, the *next bar at market* orders are evaluated. If there is more than one order with the same execution method, the order that was placed first in the strategy takes priority and is filled first.

Once all market orders are evaluated, the Trading Strategy Testing Engine analyzes stop and limit orders. If there are multiple stop or limit orders, then TradeStation gives a higher priority to the order that is closest to the market (closest to the current price). This is done in order to simulate how stop and limit orders are actually filled. If a symbol is trading at 649, and there are two limit orders to buy—one at 648 and one at 647—as the market drops, the order to buy at 648 would be filled first, and the order to buy at 647 would be filled second. Therefore, TradeStation fills these orders in that way, producing results that are as realistic as possible. If stop orders are used and you have two orders to buy at 650 and 651 and the current price is 649 and rising, then the 650 stop is filled first and then the 651. By generating exact entry and exit orders, strategies can be back tested over several years of historical data to determine effectiveness. (Chapter 5 is dedicated to the analysis of strategy performance and optimization.) The rest of this chapter will illustrate how to create trading strategies utilizing built-in EasyLanguage functions.

```
ADX and Moving Average based Strategy
{ADX by Wells Wilder—ADX determines trendiness}

Inputs: adxLength(14),mavLength1(9),mavLength2(19);
Vars: adxVal(0);
adxVal = Adx(adxLength);
if(adxVal>=15) then {we are in trending mode}
begin
       if(Average(Close,mavLength1) crosses above Average(Close,mavLength2)) then
       buy tomorrow at High stop;
       if(Average(Close,mavLength1) crosses below Average(Close,mavLength2)) then
       sellshort tomorrow at Low stop;
end;
if(adxVal<15) then {we are not in trending mode)
begin
       if(MarketPosition = 1) then Sell next bar at Lowest(Low,4) stop;
       if(MarketPosition = –1) then BuyToCover next bar at Highest(High,4) stop;
end;
```

This simple strategy incorporates the ADX and Average functions. You may have noticed the words *crosses above* and *crosses below* in these functions. These are keywords that provide a shortcut:

```
if(Average(Close,mavLength1) crosses above Average(Close,mavLength2)) then
```

is the same as

```
if(Average(Close,mavLength1)[1] < Average(Close,mavLength2)[1] and
Average(Close,mavLength1) > Average(Close,mavLength2) then
```

Not only does this save on keystrokes, it also saves on potential typos. We would suggest using these keywords whenever possible. This strategy uses the ADX indicator to determine market trend and moving average crossover to initiate positions. If the ADX determines the market is no longer trending, the system looks to exit the market with a tight stop.

Embedded Functions

```
Inputs: movAvgLength1(9),channelLength(20);
value1 = Highest(Average(Close,movAvgLength1),channelLength);
value2 = Lowest (Average(Close,movAvgLength1),channelLength);
Buy tomorrow at value1 stop;
Sellshort tomorrow at value2 stop;
```

This extremely simple system demonstrates how EasyLanguage calls a function within another function call. EasyLanguage evaluates the function calls from inside out. So, the following statement:

```
value1 = Highest(Average(Close,movAvgLength1),channelLength);
```

first calls the Average function using the closing prices for past nine days. The Highest function is then called and evaluates the past individual 20 bars' moving average values. It is hard to visualize how EasyLanguage evaluates the two functions. Table 4.1 illustrates this process used to calculate value1.

Momentum, RSI-Based Strategy with Built-in Money Management

```
Inputs: momLength(14),rsiLength(14),protStop$(3000),
trailStopThresh$(3000),trailStopPrcnt(25);
if(Momentum(Close,momLength)>0 and RSI(Close,rsiLength) crosses below 60)
then
begin
       buy("Mom+RetB")next bar at Lowest(Low,3) limit;
end;
if(Momentum(Close,momLength)<0 and RSI(Close,rsiLength) crosses above 40)
then
begin
```

```
      SellShort("Mom-RetS") next bar at Highest(High,3) limit;
end;
SetStopLoss(protStop$);
SetPercentTrailing(trailStopThresh$,trailStopPrcnt);
```

This strategy calls the Relative Strength Index (RSI) and Momentum functions. The RSI determines overbought and oversold conditions, whereas the

Table 4.1
How TradeStation Calculates the Highest Moving Average Value

Day	Close	9-Day Avg	Max. of 20 Avg. Vals
1	94.50		
2	95.25		
3	94.00		
4	93.50		
5	94.00		
6	94.25		
7	95.00		
8	95.50		
9	95.75	94.64	
10	96.00	94.81	
11	95.00	94.78	
12	95.25	94.92	
13	95.00	95.08	
14	94.50	95.14	
15	93.75	95.08	
16	92.50	94.81	
17	91.75	94.39	
18	92.25	94.00	
19	93.00	93.67	
20	93.25	93.47	
21	93.50	93.28	
22	94.00	93.17	
23	94.25	93.14	
24	94.50	93.22	
25	95.00	93.50	
26	95.50	93.92	
27	96.25	94.36	
28	96.75	94.78	95.14
29	97.00	95.19	95.19
30	97.25	95.61	95.61

Momentum function determines trend direction. The strategy buys the market when it senses a retracement from the short-term trend; momentum is positive and the RSI is exiting the overbought region. Just the opposite is true for shorting the market. Again we used the keywords, *crosses above* and *crosses below*. You will notice two new function calls: SetStopLoss and SetPercentTrailing.

SetStopLoss informs TradeStation to limit losses to the amount that is passed to the function. In our code, that amount is $3000. This is yet another shortcut that makes life a whole lot easier. You can, of course, program your own protective stops. If you are using a simple fixed money management stop, then we would suggest using this function to save programming time (well, not all of the time—in the next paragraph, we explain situations where SetStopLoss is sometimes inaccurate).

SetPercentTrailing informs TradeStation to invoke a trailing stop after a certain threshold of profit has been achieved. In our strategy, we set the threshold to $3000. Once the threshold of profit is met, TradeStation will risk a specific percentage of maximum open profit. In our snippet of code we used 25 percent. This may be easier to understand with a concrete example. Let's say we entered a long S&P500 position at 1345.00 and the market went to 1357.00 ($3000 open profit). TradeStation would then try to lock in $2250 (75% of open profit) if the market went against our profit. If the market moves higher, TradeStation will trail the high of the day in a fashion that will only risk 25 percent of total open profit. This is an effective trailing stop; the profit objective allows the market room to gyrate without premature trade termination. There is a danger involved with this built-in trailing stop if you use too tight a trailing stop percent on daily bars. This stop mechanism needs to know what occurred first—the high or the low of the day; did the market open and then move higher and pull our trailing stop up and then move down and stop us out? Or did the market first move down and stop us out and then move up and make new highs? This type of information cannot be discerned from a daily bar. We like the concept of this type of stop, but to be as accurate as possible, we would only use the SetPercentTrailing with intraday data. You can get around this problem by programming the trailing stop yourself and by trailing the stop from the high of the previous day. This would prevent the need for the chronological order of the high and low of the day. This may cause you to lose some historical back testing profit if today's high goes higher than yesterday's high and then retraces back to the trailing stop (we have prevented TradeStation from exiting on the day a new high is made), but the accuracy of testing should increase. While we are on the subject of accuracy, the SetStopLoss function also suffers from this ailment. Any time you enter and exit on a daily bar (other than entering on the open and exiting on the close), this type of error can creep into your historical analysis. Keep in mind this only applies to historical back testing on daily bars. If you back test on intraday data or trade on real-time daily bars, this type of error will not occur. We discuss this prob-

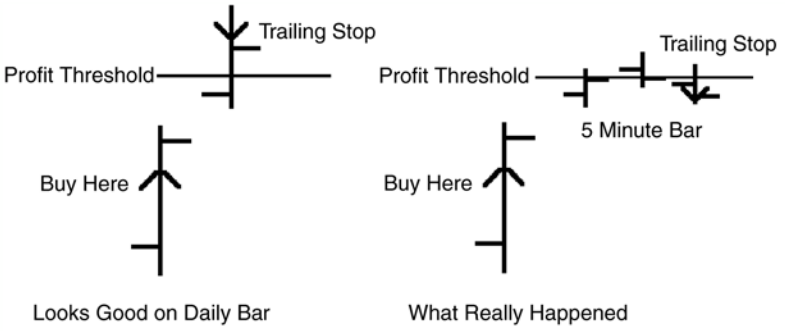

Figure 4.6 Incorrect Daily Bar Assumption

lem and TradeStation's attempt to solve it in Chapter 6. Figure 4.6 illustrates an incorrect daily bar assumption.

```
User-Defined Percent Trailing Stop
Inputs: trailStopThresh$(3000),trailStopPrcnt(25);
Vars: maxPositionProf(0),longLiqPoint(0),prevMarketPosition(0);
If(MarketPosition <>prevMarketPosition) maxPositionProf = 0;
PrevMarketPosition = MarketPosition;
if(MarketPosition = 1) then
begin
       maxPositionProf = MaxList(High – EntryPrice,maxPositionProf);
       if(maxPositionProf*BigPointValue >= trailStopThresh) then
       begin
              longLiqPoint = EntryPrice + (maxPositionProf * (1-
              trailStopPrcnt/100.0) );
              sell ("TrailLongLiq") next bar longLiqPoint stop;
       end;
end;
```

CONCLUSIONS

Several different forms of analysis techniques were discussed:

Indicator—a graphic representation of a mathematical calculation.

PaintBar and ShowMe—graphic demarcation of a mathematically-based criterion.

Function—the most important programming component in the concepts of reusable code and the dissemination of information.

Strategy—the natural conclusion of all of our programming knowledge. This is the vehicle that we use to create a mechanical trading plan.

We also touched upon the problems with back testing on daily bars. The best rule of thumb for back testing on daily bars is to prevent entries and exits on the same daily bar (unless your entry or exit is on the open or close of the bar).

The following chapters use this new knowledge to propel us into the world of trading strategies (the guts and the glory). The remainder of the book is where you will find its heart and soul. You will also find a ton of programming code. This code should reinforce what you have already learned and also take you well beyond the status of a beginning EasyLanguage programmer.

5

Measuring Trading System Performance and System Optimization

Before you can design a good trading strategy, you must first know what one is. The validity of a strategy is defined by its trading performance. In this chapter, we cover the key performance statistics that separate the good strategies from the bad. Today, with the use of TradeStation, EasyLanguage, and back-adjusted data, we can test an infinite number of different strategies on thousands of different tradable instruments. Twenty years ago only a handful of people throughout the world had this power. Power can be used or abused. In addition to performance statistics, the concepts of parameter robustness and overoptimization are discussed.

Omega Research's System Writer was one of the first software packages to provide back testing capabilities and trading performance analysis. Through the years and a couple of name changes, this software package has evolved. In its latest incarnation, back testing has become much easier to facilitate, and the number of trading performance statistics has increased ten-fold. Unfortunately, throughout the years, one key analysis tool has not found its way into TradeStation: portfolio analysis. We have the ability to microscopically analyze a trading strategy on a particular market, but we can't analyze the interaction of a strategy on a portfolio of different markets. Sure, we can have multiple tests of a strategy on different markets, but we can't see how trading several markets simultaneously will affect the bottom line. To illustrate this point, let's assume that we have developed a trading strategy and we tested it on the U.S. Bond and soybean markets. After applying the strategy to the two charts, we end up with two performance reports. From these reports, we extract the profit or loss and maximum draw-down figures (maximum draw down is the biggest $ drop in account value—actual or paper loss). To get the total

profit or loss of the two markets, we simply add these two results together. To get the maximum draw down of the two markets, you can't simply add the two values together. Draw downs for anticorrelated markets almost always occur at different times. Therefore, the overall maximum draw down of the two markets will always be equal to or less than the combined draw downs. This is the magic of diversification. When you are looking at a single market analysis, you don't see the interaction of the other markets in your portfolio. If you are actually trading the U.S. Bonds and soybeans, you know that these two markets move independently of each other. If you are losing in a bond position, you may be winning in a soybean position. In effect, offsetting a loser with a winner. Table 5.1 illustrates the magic of diversification.

We feel this is a huge oversight. Fortunately, RINA Systems, a third-party software developer, has created a solution for this problem. With their software and TradeStation, you can test a basket of different commodities or stocks and evaluate how a particular strategy performs at the portfolio level. Since we have discussed portfolio analysis with such gusto, you may be asking, "Is testing a portfolio of markets simultaneously necessary in the determination of a good trading strategy?" The answer would be no. We feel that even with this limitation, you can still develop a good strategy by testing over a well-diversified portfolio. If a strategy tests well on most of the markets in a diversified portfolio, then you may have a good strategy. If a strategy only does well in a small sector of markets that are highly correlated (markets that move somewhat in unison), then you may have an okay system that will probably generate higher draw downs. Testing markets on a simultaneous basis can give a more accurate initial capital allocation reading and open the door for in-depth money management research. Even though we can't test at the portfolio level, let's forge ahead and utilize the tools that are available.

TRADESTATION'S SUMMARY REPORT

Let's take a look at the performance statistics generated by TradeStation by plotting a chart of 2000 daily bars of the Japanese Yen and applying MyStrategy-1 (the first strategy we developed in Chapter 1). After you have applied the strategy, go under the View menu and select view strategy performance. Click on the Summary tab. A report similar to the one in Table 5.2 should show up on your screen.

This summary report has most of the statistics you need to help in your search for a good trading strategy. There is a lot of information here, but some of the statistics, in our opinion, are superfluous. We will only touch upon the most influential ones.

Table 5.1
Magic of Diversification

Strategy A	Account $Value	U.S. Bonds Draw Down	Account $Value	Soybeans Draw Down	Overall $Value	Draw Down
Jan.	$1,000	$0	$400	$0	$1,400	0
Feb.	($500)	($1,500)	$500	$0	$0	($1,400)
March	($1,000)	($2,500)	$700	$0	($300)	($1,700)
April	$250	($2,500)	$1,200	$0	$1,450	($1,700)
May	$1,000	($2,500)	$750	($450)	$1,750	($1,700)
June	$3,000	($2,500)	$250	($950)	$3,250	($1,700)
U.S Bonds Max. Draw Down		($2,500)		Soybeans Max. Draw Down		($950)
		Overall Max. Draw Down		($1,700)		

Table 5.2
Summary Performance of MyStrategy-1 on JY

TradeStation Strategy Performance Report

TradeStation Strategy Performance Report—MyStrategy-1 @ JY-Daily (4/28/94–4/11/02)

Performance Summary: All Trades

Total Net Profit	$70,075.00000	Open position P/L	($2,125.00000)
Gross Profit	$123,712.50000	Gross Loss	($53,637.50000)
Total # of trades	28	Percent profitable	39.29%
Number winning trades	11	Number losing trades	17
Largest winning trade	$28,600.00000	Largest losing trade	($5,662.50000)
Average winning trade	$11,246.59091	Average losing trade	($3,155.14706)
Ratio avg win/avg loss	3.56452	Avg trade (win & loss)	$2,502.67857
Max consec. Winners	4	Max consec. losers	6
Avg # bars in winners	124	Avg # bars in losers	33
Max intraday drawdown	($18,137.50000)		
Profit Factor	2.30646	Max # contracts held	1
Account size required	$18,137.50000	Return on account	386.35%

Performance Summary: Long Trades

Total Net Profit	($2,087.50000)	Open position P/L	($2,125.00000)
Gross Profit	$37,987.50000	Gross Loss	($40,075.00000)
Total # of trades	14	Percent profitable	28.57%
Number winning trades	4	Number losing trades	10
Largest winning trade	$13,687.50000	Largest losing trade	($5,662.50000)
Average winning trade	$9,496.87500	Average losing trade	($4,007.50000)
Ratio avg win/avg loss	2.36978	Avg trade (win & loss)	($149.10714)
Max consec. Winners	2	Max consec. losers	5
Avg # bars in winners	93	Avg # bars in losers	29
Max intraday drawdown	($20,537.50000)		
Profit Factor	.94791	Max # contracts held	1
Account size required	$20,537.50000	Return on account	–10.16%

Table 5.2
(Continued)

Performance Summary: Short Trades			
Total Net Profit	$72,162.50000	Open position P/L	$0.00000
Gross Profit	$85,725.00000	Gross Loss	($13,562.50000)
Total # of trades	14	Percent profitable	50.00%
Number winning trades	7	Number losing trades	7
Largest winning trade	$28,600.00000	Largest losing trade	($4,325.00000)
Average winning trade	$12,246.42857	Average losing trade	($1,937.50000)
Ratio avg win/avg loss	6.32074	Avg trade (win & loss)	$5,154.46429
Max consec. Winners	4	Max consec. losers	3
Avg # bars in winners	141	Avg # bars in losers	39
Max intraday drawdown	($7,712.50000)		
Profit Factor	6.32074	Max # contracts held	1
Account size required	$7,712.50000	Return on account	935.66%

Total Net Profit

This performance statistic is the brass ring that most inexperienced strategy developers reach for. You might think that a strategy that maximizes profit is the way to go. If you had Bill Gates' net worth, this would be true. Most traders have limited capital and must monitor risk at all times. What's wrong with a strategy that makes $100,000 and has a $50,000 draw down? Doesn't the end justify the means? The one question that you must ask yourself when looking at the performance of such a system is, "When did the draw down take place?" Did it happen right off the bat when you only had $10,000 in trading capital? Or did it happen after you have an additional 100K in the bank? If you don't have at least $50,000 in trading capital, then forget about it and go on with your search. Also, this statistic tells you nothing about how the profits were distributed through time. Would you trade a strategy on a market that made all of its money over a short period of time while it lost or was dormant for the rest of the test period? Another question, which is better—A strategy that makes $75,000 in the Nasdaq futures, or a strategy that makes $35,000 over the same time period in soybeans. If you were to ask us this question, we would say the soybean strategy. Sure it only made 46 percent as much as the Nasdaq system,

but it did it with a market that has one tenth or less in dollar moves. This implies a heck of a lot less risk. When you are developing strategies, use this statistic as a benchmark and not an absolute goal. Of course, you want a strategy with a positive expectation, but let's use this statistic in concert with other equally important statistics.

Maximum Intraday Draw Down

This statistic is equally important as Total Net Profit. However, the importance is somewhat diluted if you are planning on trading a diverse portfolio. Maximum draw down (or should we refer to this as meltdown) is calculated by finding the highest peak in the equity curve and subtracting the subsequent lowest trough in the curve. In other words, how much money did an account value go down before the account made a new equity high? There are two forms of maximum draw down: closed trade and open trade. Closed trade draw down occurs after a trade is closed out, and open trade draw down occurs while a trade is still on. Let's say you are trading a long-term U.S. Bond strategy and that you have a short position when the FED announces a surprise cut in interest rates. Assume the surprise rate cut causes the bond futures market to rise three full points by the end of the day. By this time your account value has dropped $3000. You have just experienced an open trade draw down of $3000. Now, let's go a little further with our hypothetical scenario. The next morning after a few hours of rumors, gossip, and information digestion, the bond market gaps down a full point and over the next few days trades back down to your entry price. The next few weeks the bond market trades in a range and you finally get out of the position with a $500 loss. You have just experienced a closed trade draw down of $500. Which figure do you think is more important? The open trade draw down is more important to the trader with $2,000 than a trader with $40,000. We feel that the maximum open trade draw down prepares a trader for the worst-case scenario. TradeStation reports maximum draw down in this manner. Total Net Profit equates to reward, whereas Maximum Draw Down equates to risk. As a strategy developer and trader, you should try to maximize the risk to reward ratio within your financial boundaries.

Account Size Required and Return on Account

You will probably notice that the Account Size Required statistic is the same as the Maximum Intraday Draw Down. Basically, TradeStation is stating that an account should be funded by the amount that the system has drawn down on a historical basis. We feel this is insufficient. The industry rule of thumb for funding a trading account is two or three times maximum draw down. You may be wondering, why double or triple the worst-case scenario to come up with the starting capital? If a system has been tested over many years, then shouldn't

the future maximum draw down fall in the range of historic parameters? This type of thinking is what makes many stock and futures traders fail within the first year of trading. You must understand that historic performance statistics have the benefit of hindsight. Any system or indicator development requires a certain level of curve fitting. Curve fitting is the testing, changing, retesting process that a developer goes through to produce a winning system. There is nothing wrong with this, unless you go overboard. Curve fitting customizes a trading idea to the historical data. The more you curve fit, the more you force history to repeat itself into the future. You may have thought the maximum draw down was the worst-case scenario, when in fact you were looking through rose-colored glasses. With this in mind, you should capitalize a trading account sufficiently to endure more than just the historic maximum draw down. Again, the lack of diversification analysis rears its ugly head. Remember when we touched upon this subject and demonstrated that overall maximum draw down can be reduced by trading diversified markets; this figure does not take diversification into consideration. If you were to trade ten different and diverse markets, should you go through and add up the individual maximum draw downs to calculate the initial account funding? We think this is overkill and a nonefficient use of capital. We know we just said that this figure alone is not sufficient enough to fund a trading account, and now we are saying that if you sum all of the maximum draw downs it would be overkill. Recall though that we did state if you were trading a diverse portfolio, this summation of maximum draw down would be inefficient use of capital. If you are trading four foreign currencies, then it may not be a bad formula of initial capital allocation. For these reasons, we feel this statistic is basically useless. The Return On Account would also fall into the useless camp.

Average Trade

This statistic is the average profit or loss that you can expect on any given trade. The key word is expect. If you trade a statistically significant number of times and take the average of your profits (or losses), then you should come close to this figure, theoretically speaking that is. You could rely more on this statistic if back testing did not incorporate the benefit of hindsight and trade results were normally distributed. Nonetheless, normal distribution statistics are what we have and we must use what we have. The importance of this statistic increases with the number of real-time trades; the assumption of normal distribution doesn't go away, but the benefit of hindsight lessens. Of course, the higher the average trade, the better. Many traders incorrectly place more emphasis on percentage of winning trades than they do on average trade. Intuitively, you may think a system that doesn't have at least 50 percent wins is a loser. This is not the case. In fact, most profitable trading systems have much less than 50 percent winners. Of the following two systems, which would you trade?

	System A	System B
First Trade	–$200	$100
Second Trade	–$150	$75
Third Trade	$400	$125
Fourth Trade	$500	$50
Fifth Trade	$1000	$25
Average Trade	$310	$75
Percent Wins	60	100

Maximum Consecutive Winners and Losers

These two statistics are used more as psychological tools than analysis tools. Consecutive losers are, from a psychological standpoint, the aspect of trading that forces most traders to call it quits. Can a typical trader sit through 15 consecutive losing trades? It doesn't matter that the trades don't add up to be a big loss. It's the loss in confidence that gnaws away at your trading plan. If you can live with historic consecutive losses plus a few more and the other performance statistics are within your risk-reward parameters, then you know that you may have a good fit with your system.

Number of Trades and Average Number of Bars Per Trade

These two statistics indicate how often a trading system trades and how long one might hold on to a position. If you like a lot of action, then you would want a system that trades frequently and gets out quickly. If you are more of a slow mover, then you would want a system that trades infrequently and holds on to a position for a longer time period. In our experience, the latter type system is usually the best. The system that trades more frequently must be more profitable due to the increase costs of trade executions. Let's say a system trades ten times a year at a cost of $50 a trade. The system must make $500 before it shows any profit. On the other hand, a system that trades five times a year with an associated cost of $50 only needs to clear $250 before it is profitable.

Average Winning and Losing Trade

These statistics inform us of the magnitude of money that we can expect to win or lose on average over an extended trading period. Can you make money with a system that has a smaller average win than average loss? Absolutely, if the percentage of winning trades is much greater than percentage of losing trades you can. Typical trading systems have less than 50 percent wins and an average winning trade that is greater than the average losing trade. The better

trend following systems will take little bites at the market until a big trend occurs. These systems may only win 30 percent of the time, but their average win is three or more times the size of their average loss. Since shorter-term systems are in the market for small amounts of time, their average wins will be much smaller than a system that holds on to a position for months at a time.

TradeStation gives almost the exact same statistics on long trades and short trades. This is potentially a nice feature. As a trader, you may want to know if you made all of your money on the long side or the short side. Let's say, hypothetically, that you were designing a trading system and discovered that you made four times as much on the long side as you did on the short side. As a system developer and potential trader, what would you do with this information? You could change your system to trade less on the short side or trade more on the long side. By doing so, you could make the historical track record look even better. The system developer side of you thinks this is great! The trader side of you doubts the validity of the optimization. Optimization? What optimization? Is it not true that you changed a parameter or two to better-fit history? You try to rationalize your decision to optimize by explaining to your trader side that history has proven itself over many years. The system developer side can't help that the market has had a bullish bias for more than eight years. The trader side gives in because it can't argue with an empirical test. So, after months of saving money for trading capital, you plunk your $50,000 down and start trading your optimized trading system. The date is January 3, 2000 and the rest is history. Of course, hindsight is 20/20—but that was what you were betting on. We feel that a trading system should be as symmetric (buy rules are just the opposite of sell rules) as possible. We know and believe that bull markets move differently than bear markets. But, we also feel that a trading system should be equipped for any possible circumstances. So, use this information along with the other statistics. Don't change your trading system or style to try and take advantage of any divergence between long and short profits.

TRADES

If you click on the Trades tab at the bottom of the Performance Report window, a different window will pop up giving you trade-by-trade statistics. You can choose standard view or advanced. First let's take a look at the advanced view. (You can do this by selecting Adv. from the top left combo box.) Your screen should look somewhat similar to Table 5.3.

Every trade statistic that you could possibly want is given here: signal, entry date and price, exit date and price, profit/loss, percent profit, cumulative profit, commission, slippage, run-up, draw down, entry and exit efficiency, and total efficiency. The key statistics in this report are individual trade run-up and draw down, entry and exit efficiencies, and total trade efficiency.

Table 5.3
Advanced Trade by Trade Analysis of MyStrategy-1

Trade # Typ	Date	Time	Price	Signal	Contracts Profit	% Profit Cum Profit	Commiss.	Slippage	Run-up Draw Down	Entry Eff. Exit Eff.	Total Efficiency
1	10/10/94	00:00	1.32860	Short	1	(2.60%)	$0.00	$0.00	12.50	0.29%	
Sell	10/19/94	00:00	1.36320	Buy	(4325.00)	(4325.00)	0.00	0.00	(4325.00)	(0.00%)	(99.71%)
2	10/19/94	00:00	1.36320	Buy	1	(2.55%)	$0.00	$0.00	1400.00	24.40%	
Buy	12/2/94	00:00	1.32850	Short	(4337.50)	(8662.50)	0.00	0.00	(4337.50)	(0.00%)	(75.60%)
3	12/2/94	00:00	1.32850	Short	1	(1.69%)	$0.00	$0.00	1875.00	40.11%	
Sell	2/16/95	00:00	1.35090	Buy	(2800.00)	(11462.50)	0.00	0.00	(2800.00)	(0.00%)	(59.89%)
4	2/16/95	00:00	1.35090	Buy	1	7.48%	$0.00	$0.00	26000.00	97.97%	
Buy	7/10/95	00:00	1.45190	Short	12625.00	1162.50	0.00	0.00	(537.50)	49.60%	47.57%
5	7/10/95	00:00	1.45190	Short	1	15.76%	$0.00	$0.00	33525.00	97.56%	
Sell	4/29/96	00:00	1.22310	Buy	28600.00	29762.50	0.00	0.00	(837.50)	85.67%	83.23%
6	4/29/96	00:00	1.22310	Buy	1	(3.59%)	$0.00	$0.00	50.00	0.90%	
Buy	5/28/96	00:00	1.17920	Short	(5487.50)	24275.00	0.00	0.00	(5487.50)	0.00%	(99.10%)
7	5/28/96	00:00	1.17920	Short	1	(0.68%)	$0.00	$0.00	3125.00	60.68%	
Sell	7/31/96	00:00	1.18720	Buy	(1000.00)	23275.00	0.00	0.00	(2025.00)	19.90%	(19.42%)

8	7/31/96	00:00	1.18720	Buy	1	(2.57%)	$0.00	$0.00	500.00	11.59%	
Buy	9/10/96	00:00	1.15670	Short	(3812.50)	19462.50	0.00	0.00	(3812.50)	(0.00%)	(88.41%)
9	9/10/96	00:00	1.15670	Short	1	9.71%	$0.00	$0.00	19287.50	95.72%	
Sell	5/9/97	00:00	1.04440	Buy	14037.50	33500.00	0.00	0.00	(862.50)	73.95%	69.67%
10	5/9/97	00:00	1.04440	Buy	1	2.36%	$0.00	$0.00	8575.00	87.72%	
Buy	7/15/97	00:00	1.06910	Short	3087.50	36587.50	0.00	0.00	(1200.00)	43.86%	31.59%
11	7/15/97	00:00	1.06910	Short	1	8.33%	$0.00	$0.00	17287.50	90.93%	
Sell	1/27/98	00:00	.98000	Buy	11137.50	47725.00	0.00	0.00	(1725.00)	67.65%	58.58%
12	1/27/98	00:00	.98000	Buy	1	(3.23%)	$0.00	$0.00	2050.00	34.10%	
Buy	3/16/98	00:00	.94830	Short	(3962.50)	43762.50	0.00	0.00	(3962.50)	(0.00%)	(65.90%)
13	3/16/98	00:00	.94830	Short	1	6.25%	$0.00	$0.00	13512.50	93.03%	
Sell	9/1/98	00:00	.88900	Buy	7412.50	51175.00	0.00	0.00	(1012.50)	58.00%	51.03%
14	9/1/98	00:00	.88900	Buy	1	12.32%	$0.00	$0.00	22637.50	96.23%	
Buy	2/16/99	00:00	.99850	Short	13687.50	64862.50	0.00	0.00	(887.50)	61.96%	58.18%
15	2/16/99	00:00	.99850	Short	1	2.15%	$0.00	$0.00	8212.50	74.83%	
Sell	7/22/99	00:00	.97700	Buy	2687.50	67550.00	0.00	0.00	(2762.50)	49.66%	24.49%
16	7/22/99	00:00	.97700	Buy	1	7.03%	$0.00	$0.00	14462.50	93.46%	
Buy	1/7/00	00:00	1.04570	Short	8587.50	76137.50	0.00	0.00	(1012.50)	62.04%	55.49%

Individual trade run-up is the total profit amount of your position before the trade was closed out. Individual trade draw down is the total amount that your position was against you before it was closed out. If you notice that your actual profit is considerably less than your run-up profit, then you know that you are leaving too much on the table. You should work on an exit mechanism that locks into a larger portion of the larger run-ups. The draw down figure gives insight into how much the market may move against an eventual profitable trade. In addition, if the draw down figures are too large on an individual basis for you to stomach, then you may want to move on to another system. These two statistics are also known as favorable and adverse trade excursion, respectively. Many people feel these statistics are key in the development of trade management techniques. The entry, exit, and total efficiency statistics are all based on the favorable and adverse excursions. The entry efficiency measures how close the entry price was to the best possible entry price during the trade. If a trade only goes slightly in your favor and then closes out with a sizable loss, then your entry efficiency will be at a low level (entry/exit efficiencies range between 0% and 100%). The exit efficiency measures how close the exit price was to the best possible exit price during the trade. If a big winner turns into a loser or small winner, then this efficiency will also be at a low level. The total efficiency measures the amount of profit captured from the total extremes of the market during the trade duration. This number can be either positive or negative, based on the result of the trade. According to our numbers on MyStrategy-1 on the Japanese Yen, our approach is not very efficient. This can be attributed to the lack of a money management plan. Nonetheless, it is still overall quite profitable. All of these statistics can give in-depth insight into your trade management methodology. For further information on trade-by-trade excursions and efficiencies, we highly recommend John Sweeney's, "Maximum Adverse Excursion: Analyzing Price Fluctuations for Trading Management" (John Wiley & Sons, Wiley Trader's Advantage Series, 1997). This report will help you verify your translation of trading idea into a mechanical system. If you know that a trade should occur on a certain day at a certain price and it doesn't show up in the trade-by-trade report, then you know that you haven't accurately programmed your ideas.

ANALYSIS

Click on the Analysis tab and you will be presented with a report like the one in Table 5.4. This report is a statistician's dream. Every statistic under the sun is located in this report. You will recognize many of the statistics from the Summary report. Again, we will only go over the more relevant ones.

The profit factor is calculated by simply dividing the gross profit by the gross loss. This factor tells us how many dollars you can expect to win for every

Table 5.4
Analysis of MyStrategy-1

TradeStation Strategy Performance Report

TradeStation Strategy Performance Report - MyStrategy-1 @JY-Daily (4/28/94-4/11/02) (4/28/94-4/11/02)

STRATEGY ANALYSIS

Net Profit	$70,075.00000	Open Position	($2,125.00000)
Gross Profit	$123,712.50000	Interest Earned	$202.73973
Gross Loss	($53,637.50000)	Commission Paid	$0.00000
Percent profitable	39.29%	Profit factor	2.31
Ratio avg. win/avg. loss	3.56	Adjusted profit factor	1.30
Annual Rate of Return	21.72%	Sharpe Ratio	0.39
Return on Initial Capital	350.38%	Return Retracement Ratio	1.29
Return on Max. Drawdown	280.02%	K-Ratio	2.61
Buy/Hold return	–44.47%	RINA Index	16.62
Cumulative return	350.38%	Percent in the market	97.45%
Adjusted Net Profit	$19,765.27320	Select Net Profit	$41,475.00000
Adjusted Gross Profit	$86,411.77778	Select Gross Profit	$95,112.50000
Adjusted Gross Loss	($66,646.50459)	Select Gross Loss	($53,637.50000)

(Continues)

Table 5.4
(Continued)

TOTAL TRADE ANALYSIS					
Number of total trades	28				
Average trade	$2,502.67860		Avg. trade ± 1 STDEV	$10,990.41	/ ($5,985.05449)
1 Std. Deviation (STDEV)	$8,487.73309		Coefficient of variation	339.15%	
Run-up					
Maximum Run-up	$33,525.00000		Max. Run-up Date	4/11/96	
Average Run-up	$7,864.22410		Avg. trade ± 1 STDEV	$17,029.16222	/ $0.00000
1 Std. Deviation (STDEV)	$9,164.93812		Coefficient of variation	116.54%	
Drawdown					
Maximum Drawdown	($6,500.00000)		Max. Drawdown Date	8/8/94	
Average Drawdown	($2,560.34480)		Avg. trade ± 1 STDEV	($947.42241)	/ ($4,173.26719)
1 Std. Deviation (STDEV)	$1,612.92239		Coefficient of variation	63.00%	
Reward/Risk Ratios					
Net Prft/Largest Loss	12.38		Net Prft/Max Drawdown	10.78	
Adj Net Prft/Largest Loss	3.49		Adj Net Prft/Max Drawdown	3.04	
Outlier Trades	**Total Trades**	**Profit/Loss**			
Positive outliers	1	$28,600.00000			
Negative outliers	0	$0.00000			
Total outliers	1	$28,600.00000			

EFFICIENCY ANALYSIS					
Total Efficiency					
Average Total Efficiency	–16.31%		Avg. trade ± 1 STDEV	47.24%	/ –79.86%
1 Std. Deviation (STDEV)	63.55%		Coefficient of variation	389.63%	
Entry Efficiency					
Average Entry Efficiency	57.12%		Avg. trade ± 1 STDEV	92.47%	/ 21.76%
1 Std. Deviation (STDEV)	35.36%		Coefficient of variation	61.91%	
Exit Efficiency					
Average Exit Efficiency	26.58%		Avg. trade ± 1 STDEV	57.95%	/ –4.80%
1 Std. Deviation (STDEV)	31.37%		Coefficient of variation	118.05%	

OPEN POSITION ANALYSIS			
	Open Position	**Average Trade**	**Percent of Average**
Unrealized Profit/Loss	($2,125.00000)	$2,502.67860	–184.91%
Time in trade (Days)	36.00	99.82	36.06%

dollar you lose. Since the profit factor is a ratio, you can compare this statistic between other markets. Other statistics, like total profit and maximum draw down, cannot be directly compared with the same statistics from different markets. When you compare the results of a system tested on the NASDAQ with the results of the same system tested on the soybean market, you are comparing apples to oranges. The profit factor is normalized between different markets. When evaluating a trading strategy performance across a broad spectrum of markets, we would skip the total profit as the only guiding factor for portfolio selection. We would use the profit factor to compare one market equally against another.

The adjusted profit factor is an even better performance statistic. Adjusted profit factor is calculated by dividing adjusted gross profit by adjusted gross loss. Adjusted gross profit/loss is calculated by subtracting/adding the square root of winning/losing trades from the total number of winning/losing trades and multiplying the result by the average winning/losing trade, that is, deflating profit and inflating loss. Why do this you may ask? Simulated analysis of trading will never accurately match real-time trading. Also, there are instances when you get an outlier trade that skews the performance statistic. If you scroll down the Analysis report, you will find the Outlier Trade analysis. Outlier trades are ones that are several standard deviations away from the average winning or average losing trade. These trades usually occur out of pure luck and, therefore, shouldn't be built into our expectations. In the case of MyStrategy-1, you will notice one $28,600 win. Should you count on this happening again? If you can cut your expectations down, then you can approach trading from a less than best-case scenario. We like to approach a new trading strategy from a worst-case perspective.

A trading strategy should provide consistency in equity growth. Who would trade a strategy that makes all of its money on a small number of trades? The *Sharpe ratio* describes a strategy's consistency. Subtracting the risk-free interest from the average monthly return and dividing by the standard deviation of the monthly returns calculates the official Sharpe ratio. The higher the ratio, the more consistent your returns. Again, this is a normalized statistic and can be used to compare different strategies or markets. The *K-Ratio* is similar to the Sharpe ratio in that it measures consistency by using linear regression techniques and, therefore, has a different scale factor.

The *Rina Index* is a proprietary calculation that takes net profit, average draw down, and percent time in market into consideration. A Rina Index above 30 usually points to a productive trading strategy. This index comes from the same company (Rina Systems) that offers the third-party add-on for portfolio analysis. Their software also helps with performance analysis.

Did you beat the old buy and hold routine? This Buy and Hold statistic is also given, but it really isn't an informative statistic outside of the world of stocks or stock indices. The Total Trade Analysis section of this report also

gives some good information. You can find out the standard deviations of the average trade, average run-up, and average draw down. These figures may more accurately guide you in allocating initial trading capital. All of TradeStation's reports can be saved in Microsoft Excel format. This is a great feature for two reasons. First, the general public can open and read the reports without owning TradeStation. Secondly, Microsoft Excel opens another enormous library of analytical tools that can be used with the reports.

GRAPHS

There are other reports that we haven't discussed that you can select by clicking on their respective tabs. We feel that all TradeStation users should familiarize themselves with these reports. We could write an entire book on the information that is presented in this research, but we really don't want to bore you more than you already are. Sometimes you can research an idea to death. The last report that we will discuss is the Graphs report. This report along with the Summary, Analysis and Trades should give you all of the information that you need to determine the validity and robustness of a trading system. Go ahead and click on the Graphs tab. A chart similar to Figure 5.1 should now appear on your screen.

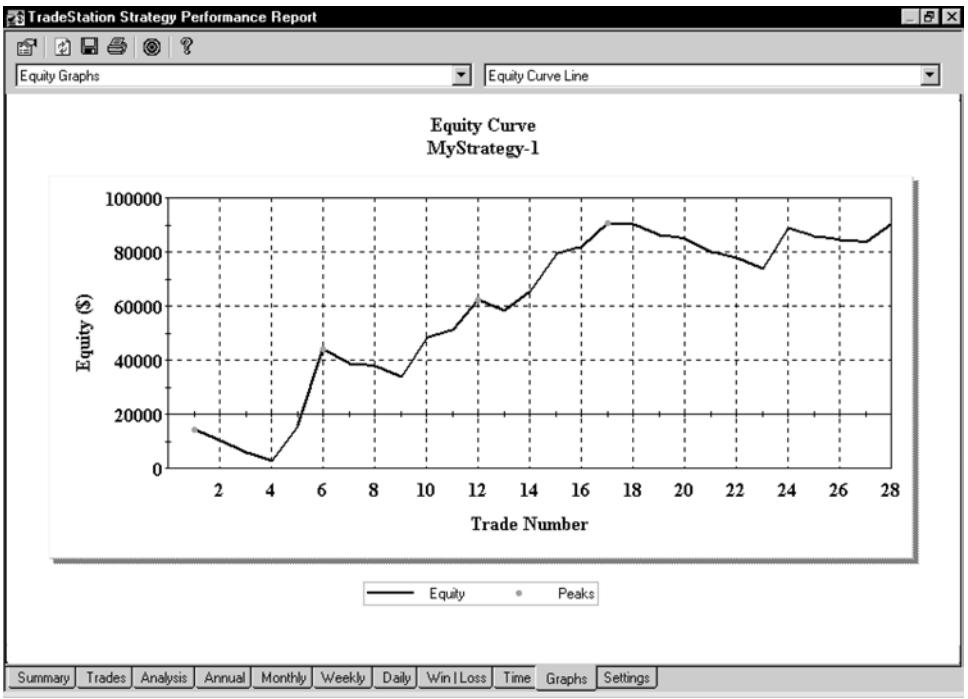

Figure 5.1 Equity Curve of MyStrategy-1

Again, there is a plethora of different types of information at your disposal. We will confine our discussion to equity graphs. The graph in Figure 5.1 illustrates the distribution of returns over the test period. This curve doesn't look that bad; it is somewhat smooth and sloping upward. The equity scale on the left starts out at $100,000, but can be changed by clicking on the Format window icon in the top right corner of the window (the one with the hand in it). You can change the format of all the reports by clicking on this icon and changing the parameters from the dialog box. Go ahead and click on this icon. A dialog box similar to the one in Figure 5.2 should pop up on your screen.

Go to the Initial Capital field and change the setting from $100,000 to $0. While we are in here, we may as well change the number of Decimal Places. Let's change this setting to 5. We did this so that we can get accurate prices in our Trades report. Click on OK and we should be back to our equity graph. Our changes may not have been reflected in the graph, so go ahead and click on the refresh icon (the icon with the two small arrows). You should now see our equity graph starting out at around $0.

Let's change the type of Equity Graph. Click on the right down arrow and select Detailed Equity Curve. The first curve shows cumulative profit on a trade-by-trade basis. The second equity curve shows cumulative profit on a daily basis. You can see the real intratrade draw downs with this graph. We like this type of equity curve best. Go back up to the right down arrow and select Underwater Equity Curve. A graph similar to Figure 5.3 should be on your screen.

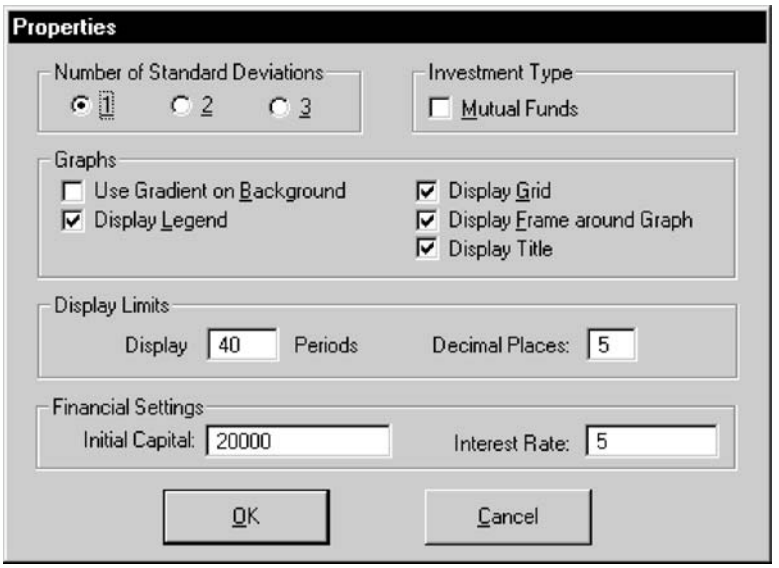

Figure 5.2 Properties of Performance Reports

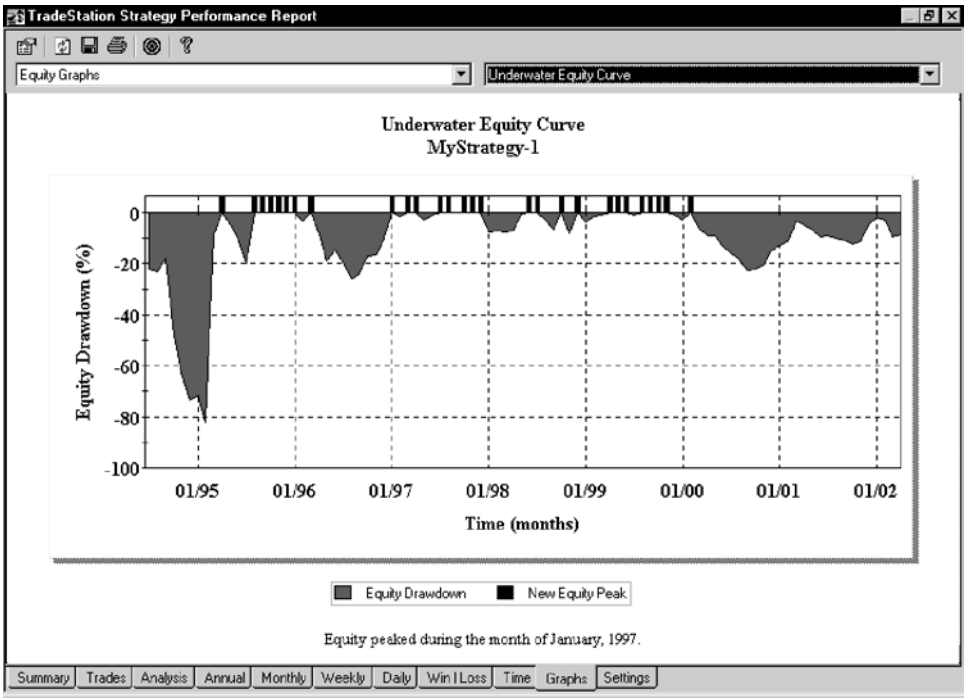

Figure 5.3 Underwater Equity Curve of MyStrategy-1

Before we explain the meaning behind this graph, let's go back to the Format window icon and change the initial capital back to $20,000. Remember to hit the refresh icon (the icon with the two arrows). This graph plots the draw downs that were experienced by our strategy during the life of the test. As you can see from the graph, we had some substantial periods of losses. This graph can really open your eyes to risk. From this graph, we can see that the strategy nearly blows us out of the water from the beginning. From that point on, we had several draw downs of differing levels. On average, it looks like we had draw downs equal to 20 percent of our initial capital. Once again, click on the right down arrow and select Monthly Net Profit. Another chart like the one in Figure 5.4 will fill your screen.

This chart plots the net monthly profits during the life of our test. Again, this may be useful information to help you decide on a strategy's validity and/or fund allocation. You may want to investigate the other graphs and charts. You may come across one that you like better than the ones that we have presented.

We hope this brief introduction to trading performance measurements has given you a base education to build on. We feel that with this education, you are prepared to differentiate between a good strategy and one that is not so good. There are several books written on this subject and we would recom-

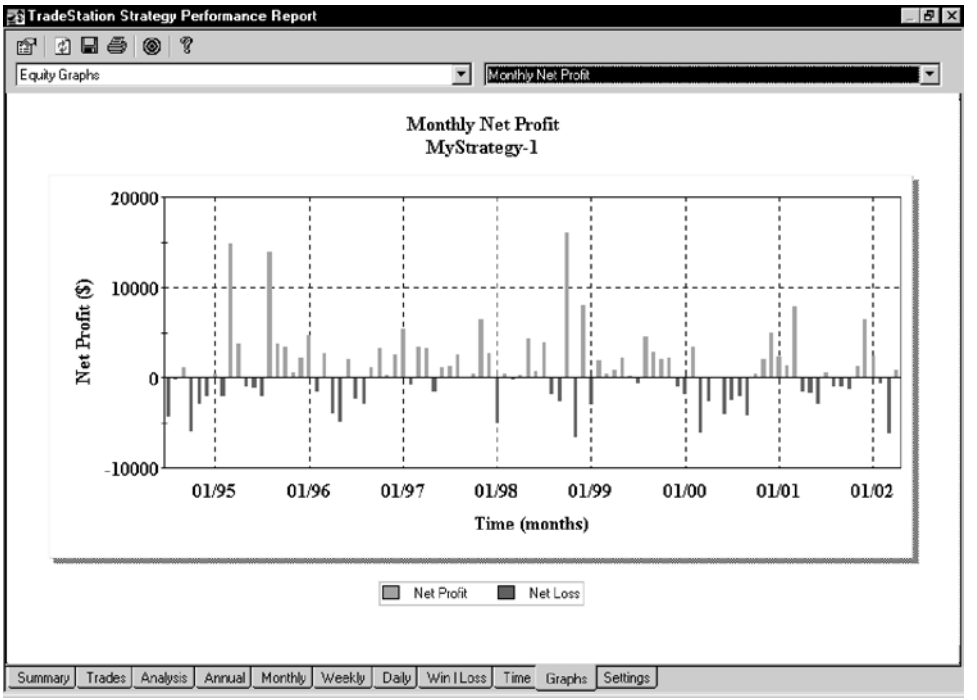

Figure 5.4 Monthly Net Profits of MyStrategy-1

mend reading most of them. Tushar Chande's and Thomas Stridsman's books should be in every traders library.

OPTIMIZATION

Now that we know some of the statistics that are used in trading system performance, let's see if we can make MyStrategy-1 a better strategy through the use of optimization. You may be asking, "What is optimization?" *Optimization* is the process of finding the optimum parameter through repeated testing of a strategy using different parameter values. In MyStrategy-1, we have two input variables: longLength and shortLength. These variables determine our exact buy and sell points. Recall when we first programmed MyStrategy-1 in Chapter 1. The strategy goes long when the high of today exceeds the high of the past longLength days and goes short when the low of today exceeds the low of the past shortLength days. We set these inputs to 40 (a somewhat arbitrary value, picked without doing much research). The optimization process will automatically change our longLength and shortLength variables (within the boundaries that we set) and re-run the strategy on the different set of parameters. The end result will be a report of the different tests on the different para-

meters. It is up to us to pick the parameter set that we eventually use. TradeStation offers a simple way to optimize a trading strategy. However, you must prepare your strategy for optimization. When initially programming a strategy, you may want to keep in mind the variables that you may want to optimize in the future. We did this with MyStrategy-1 by making the two optimizable parameters (longLength and shortLength) Input variables. Input variables are the only ones that can be optimized. Let's go ahead and optimize MyStrategy-1 on the Japanese Yen.

Make sure that the Japanese Yen continuous chart with MyStrategy-1 applied to it is active. If it is not open, just go ahead and recreate a Japanese Yen continuous chart with at least 2000 days of history. After the chart appears on your screen, apply MyStrategy-1 to it. Go under the Format menu and select Format Strategies. A dialog box that looks like Figure 5.5 will be on your screen.

Click on the Inputs button and another dialog box title *Inputs* will open. It will be similar to the one in Figure 5.6.

Click on longLength and then click the optimize button and yet another dialog box will open. Don't blame us, this is the true benefit of a graphical user interface (GUI)-driven application. The GUI allows you to make changes through an infinite number of dialog boxes. You should have a dialog box titled *Optimize* as the top window on your screen. You should see the number

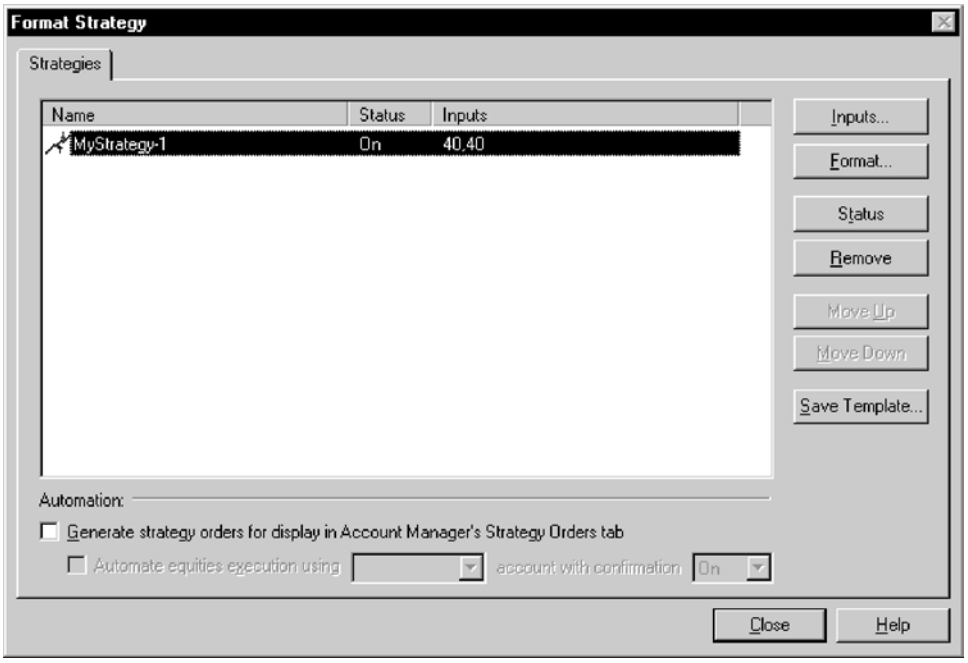

Figure 5.5 Format Strategies on MyStrategy-1

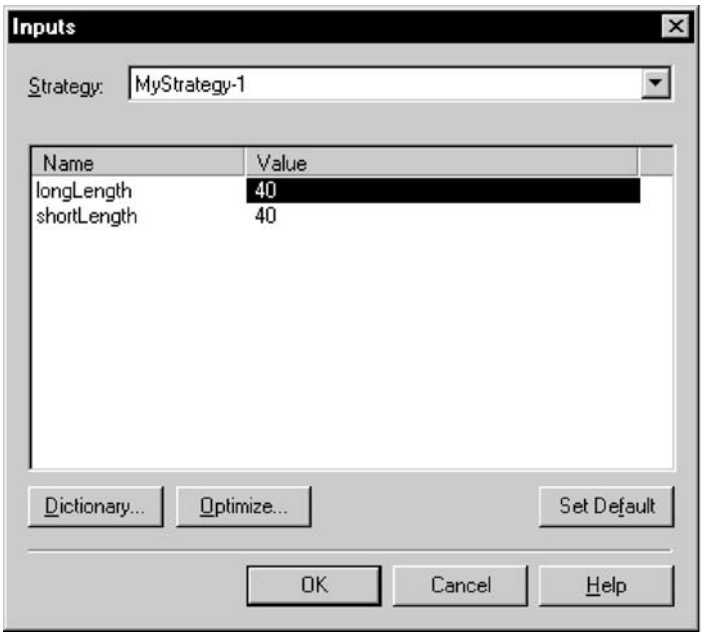

Figure 5.6 Inputs of MyStrategy-1

20 in the Start box, the number 60 in the Stop box, and the number 4 in the Increment box. Basically, this dialog box is telling us that TradeStation will repeatedly test the longLength parameter starting at the value of 20 and incrementing it by 4 until it reaches 60. The first test would set longLength to 20 and the next test would set longLength to 24 and so on. Use the following formula to determine the number of tests or iterations: (Stop – Start)/Increment + 1. By substituting our values into the formula, we come up with (60 – 20)/4 + 1 = 11. The number of tests is more important than you may think. Remember, we have two optimizable parameters. How many tests will be necessary to test both parameters simultaneously over our optimization range? Did you say 121 (11×11)? This is why the number of tests is important. The total number of tests increases in a geometric fashion. If we had three parameters and wanted to test over the same optimization range, it would be a total of 11×11×11 tests. Even with today's computing power, this could take a while. Go ahead and click OK and select the shortLength parameter and click the optimize button. You know the routine. Set this parameter optimization range to be the same as longLength. Click OK to get out of the *Input* dialog box. You should be back to the *Format Strategy* dialog box. Gentlemen start your engines. Let's get optimizing by clicking on the Optimize button. You may get an alert that says something like you tried to reference too many bars back. Basically, you crashed in turn number one. The whole optimization run is halted and you must go up

under the Format menu and select Format Strategy. Once the all too familiar dialog box comes up (you may have to click the Status button to activate the strategy and then you will need to click on the Format button). In the resulting dialog box, you will see *Maximum number of bars study will reference*. Change this to 60, so that we have enough historic data to do our optimization. Click OK and go through changing optimization values for the inputs. After you have done this, go ahead and hit the Optimize button again. It would be great if TradeStation handled this for you, but it doesn't. Once everything is set up properly and we are in the optimization process, TradeStation will keep us informed of the status of the process. It tells us the amount of elapsed time, estimated completion time, total number of runs, and current run number. It also tells us the maximum profit attained by optimizing the parameters. The optimization that we prescribed for MyStrategy-1 shouldn't take that long. Once completed, go under the View menu and select Strategy Optimization Report. This will create a spreadsheet like the one in Table 5.5.

This optimization report gives the statistics on each and every run. If you scroll down to the bottom of the report, you will notice 121 individual runs. The optimization process is quite simple compared to the search-for-robust-parameters process. In our optimization report, there is a ton of information to filter through. Should we pick the parameter set that has the most profit or smallest draw down or highest profit factor? Unfortunately, there is no black-and-white rule for picking the best parameters. Uncovering the best parameter set can be an exhaustive process of elimination. This process involves sorting, graphing, and the eventual elimination of underperforming parameters. Throughout the years of testing and optimizing trading systems, we have discovered that comparing the profit to draw down ratio or the return on account across the optimization range is the first step in the parameter selection process. So with this in mind, let's click on the floppy disk and save our optimization report to the C: drive. You may want to name the file MyStrategyOpt. Temporarily shrink TradeStation down. The next few steps will require spreadsheet software such as Microsoft Excel. We prefer Excel, and we will use it in the next few illustrations.

Launch Excel and open MyStrategyOpt, or whatever you called it, from the C: drive. TradeStation uses a delimiter to separate the fields in our optimization report. When Excel asks you to choose the file type that best describes your data, click on the Delimited radio button. Continue clicking on the Next button until you have the file opened in a spreadsheet. You will notice that our optimization report looks the same as it did in TradeStation. You might even be asking, "Why did we go through this additional step?" Microsoft Excel or any other spreadsheet program has many more capabilities when it comes to data processing than TradeStation. As we all know, a picture is worth a thousand words. The spreadsheet that we have before us provides an

Table 5.5
Strategy Optimization Report

longLength	shortLength	NetPrft	GrossP	GrossL	#Trds	%Prft	#WTrds	#LTrds
20	56	13075	105537.5	–92462.5	46	32	15	31
20	60	13775	102712.5	–88937.5	44	34	15	29
20	52	22450	110187.5	–87737.5	46	32	15	31
20	32	29575	123825	–94250	51	35	18	33
20	48	29975	114162.5	–84187.5	46	34	16	30
20	24	35200	129987.5	–94787.5	57	36	21	36
20	40	38075	120312.5	–82237.5	48	37	18	30
20	28	42350	132400	–90050	53	35	19	34
20	44	38675	117775	–79100	46	39	18	28
20	36	43050	121300	–78250	48	37	18	30
20	20	56087.5	142900	–86812.5	60	38	23	37
24	60	42075	103412.5	–61337.5	34	38	13	21
40	60	43737.5	103950	–60212.5	26	34	9	17
32	60	45287.5	106112.5	–60825	28	32	9	19

28	60	45550	104225	–58675	30	36	11	19
24	56	46725	106600	–59875	34	38	13	21
36	60	47225	104800	–57575	26	34	9	17
40	56	48387.5	106162.5	–57775	26	38	10	16
32	56	49937.5	108325	–58387.5	28	35	10	18
28	56	50200	106887.5	–56687.5	30	40	12	18
36	56	51875	107012.5	–55137.5	26	38	10	16
24	52	56100	112600	–56500	34	41	14	20
32	24	70512.5	136062.5	–65550	37	35	13	24
40	52	57762.5	111062.5	–53300	26	38	10	16
32	52	59312.5	113300	–53987.5	28	39	11	17
28	24	70775	134787.5	–64012.5	39	35	14	25
28	52	59575	112075	–52500	30	43	13	17
36	24	71950	134625	–62675	35	34	12	23
24	48	63600	118412.5	–54812.5	34	41	14	20
24	24	73575	135425	–61850	43	34	15	28

overabundance of information; our minds are not able to manipulate the information into a readily understandable format. This is where Excel comes in handy. We have now optimized two parameters on MyStrategy-1 and created several different statistics on each individual run. Since we optimized only two parameters, we will be able to create a three-dimensional contour chart of our data. Remembering back to high school math, we know that a three-dimensional chart requires three axes. In our graph, the longLength parameters will be the x-axis, shortLength parameters will be the y-axis, and the return on account will be the z-axis. (We could create different charts by using different performance statistics for our z-axis.) You may feel that the draw down statistic is the better key in parameter selection. Before we can create our chart, we must manipulate the data into a format that Excel can easily interpret and the best way to do this is by creating a pivot table. A pivot table is simply a customizable table that rearranges our data into a more readable format. If you don't have Excel, your spreadsheet software should be able to create a pivot table. (We refer you to your spreadsheet user's guide.) From within Excel, go up under the Data menu and select PivotTable and PivotChart Report. The PivotTable Wizard will open and ask you several questions concerning our data. A dialog box titled *PivotTable and PivotChart Wizard* should open and look similar to Figure 5.7.

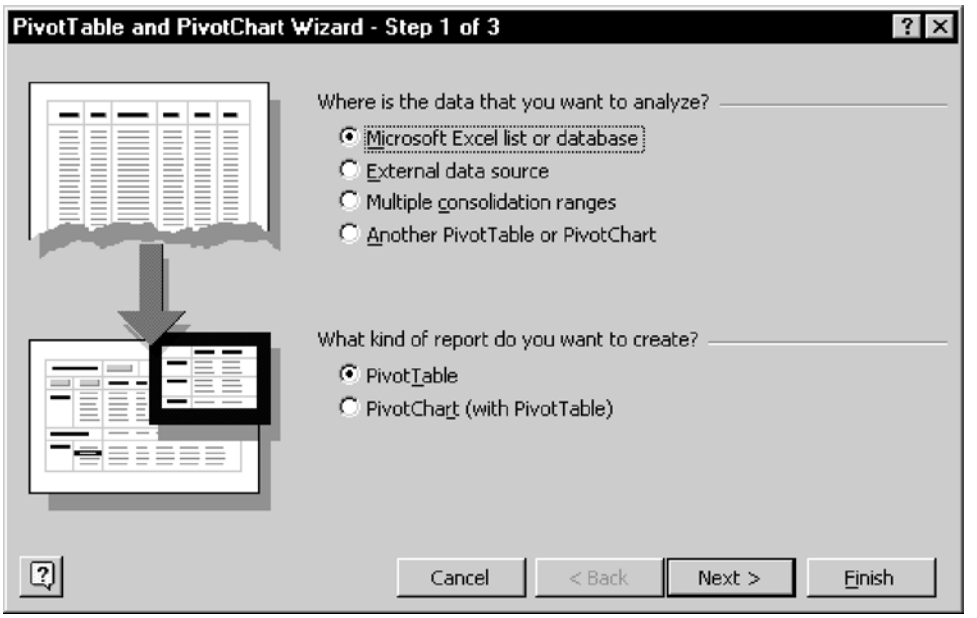

Figure 5.7 PivotTable and PivotChart Wizard—Step 1 of 3

Figure 5.8 PivotTable and PivotChart Wizard—Step 2 of 3

In the top portion of the dialog box click the *Microsoft Excel list or database* radio button. In the lower half of the dialog box click the *PivotChart (with Pivot Table)* radio button. Go ahead and click Next. The dialog box like the one in Figure 5.8 will open with the range of data that will be used in the creation of our table and chart.

Upon clicking Next you will see another dialog box. It asks if we want to put our pivot table in a new worksheet or the existing worksheet. Go ahead and put it into a new worksheet. Now this is where it gets kind of confusing. Click on the Layout button and a dialog box like the one in Figure 5.9 will open.

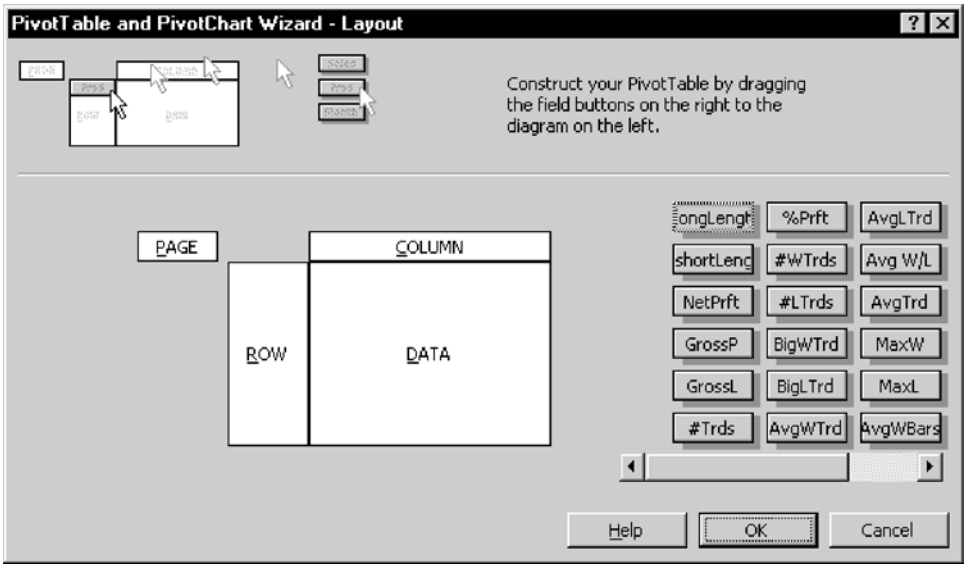

Figure 5.9 PivotTable and PivotChart Wizard—Layout

This dialog box will allow us to choose only the data we want in our table and chart. In addition, we can customize the format of our table. In the right side of the dialog box you will see all of the different statistics that TradeStation generated in our optimization report. Take a look at all of the buttons and familiarize yourself with all of the different options. You may need to use the scroll bar to see all of the available buttons. You will see a button labeled longLength. Click on it and drag it over to the area labeled ROW. Click on the button labeled shortLength and drag it over the area labeled COLUMN. Click on the button labeled ROA (you may need to use the scroll bar underneath our field buttons) and drag it to the area labeled DATA. Your dialog box should now look like Figure 5.10.

Click OK and you will be back at the previous dialog box. Click Finish. You should see several things on your screen at this point. If everything went as planned, your screen should look like Figure 5.11. (If your screen doesn't look like Figure 5.11, then start again and refer to your spreadsheet software manual.)

We have now created a pivot table chart out of our specified data. You may think, "Big deal!" The data still isn't visually pleasing. That is because we are looking at our three-dimensional data in two dimensions. Click on the chart icon near the top of the *PivotTable* window. We need to tell Excel to plot our data in three dimensions. After clicking the chart icon a dialog box like the one in Figure 5.12 will now be on your screen.

You will notice that there are several different chart types at your disposal. Click on the Surface chart selection. The surface chart has four different subtypes. Accept the default subtype and click Finish. We could click on

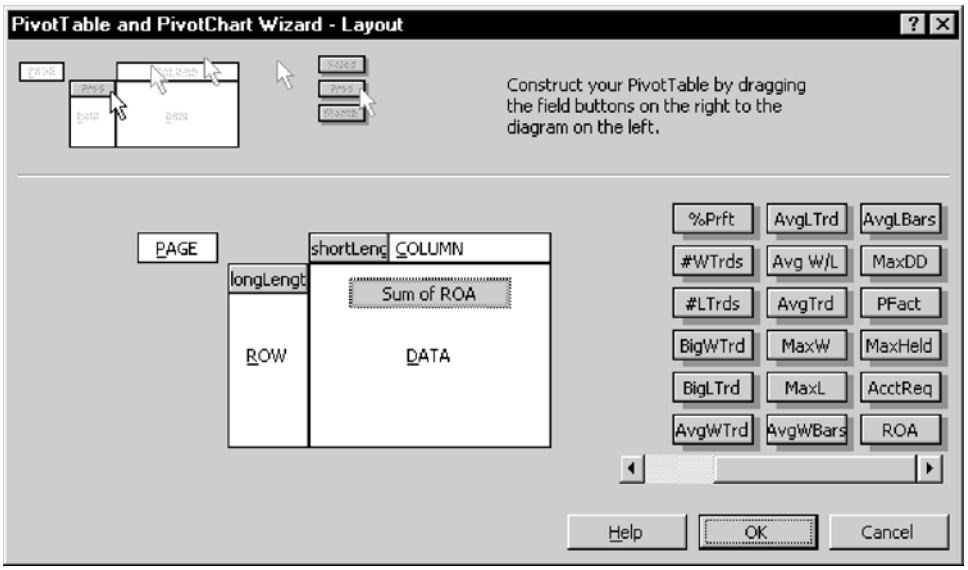

Figure 5.10 PivotTable and Selected Fields

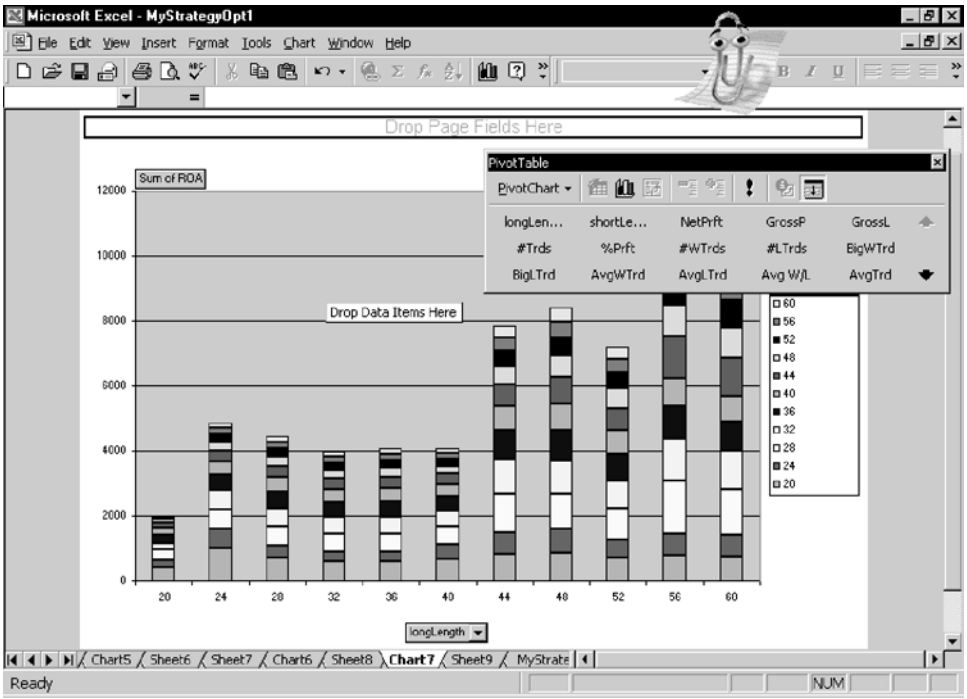

Figure 5.11 3-D Data Through the Eyes of a 2-D Chart

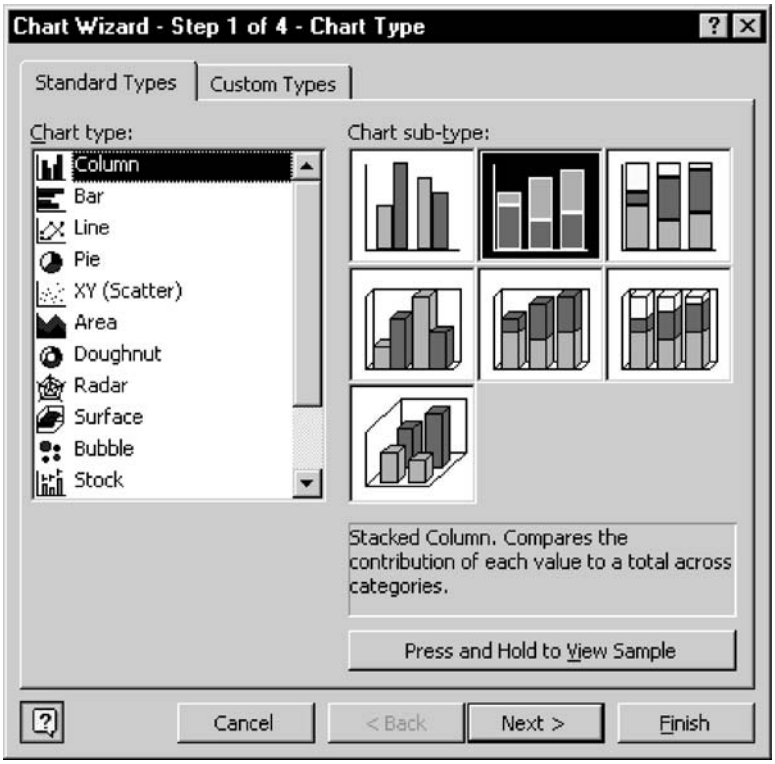

Figure 5.12 Chart Wizard

Next and then have the ability to give titles to our chart and axes, but to save time, we will simply skip this step. Voila! You should now have a three-dimensional surface or contour chart on the screen. Figure 5.13 shows the end result of our diligent efforts.

If you are unfamiliar with these types of charts, you may be asking, "What the heck am I looking at?" A surface chart helps to determine the optimum combinations between two sets of data. The x and y axes are our longLengths and our shortLengths respectively, and the z-axis is the Return on Account. The chart looks like a mountain range; there are peaks, valleys, and plateaus. Parameters that fall on the peaks are not considered robust, even though they produced the best profit/draw down ratio. Also parameters that fall in valleys are not considered robust because they did not produce acceptable profit/draw down ratios. The most robust parameters fall on plateaus. Plateau parameters do not require history to repeat itself exactly to produce similar historic results, whereas peak parameters do. Let's say the combination of 56 and 28 produced the highest return and the highest peak on the chart. Did the parameters around this peak have a dramatic drop off in performance? You bet they did. This tells us that if history doesn't repeat itself exactly, then we cannot expect our tests to approximate future performance. From the surface chart, we feel the combination of parameters that fall in the range of [longLength 40–48] and [shortLength 40–48] would produce the most robust results. These parameters

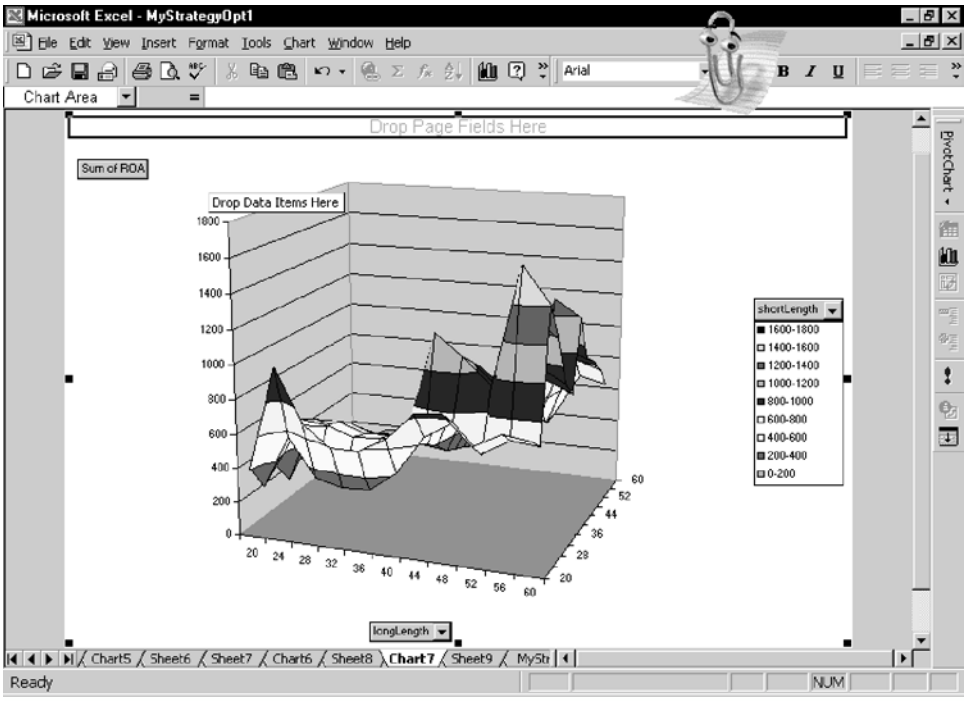

Figure 5.13 3-D Contour Chart of MyStrategy-1 Optimizations

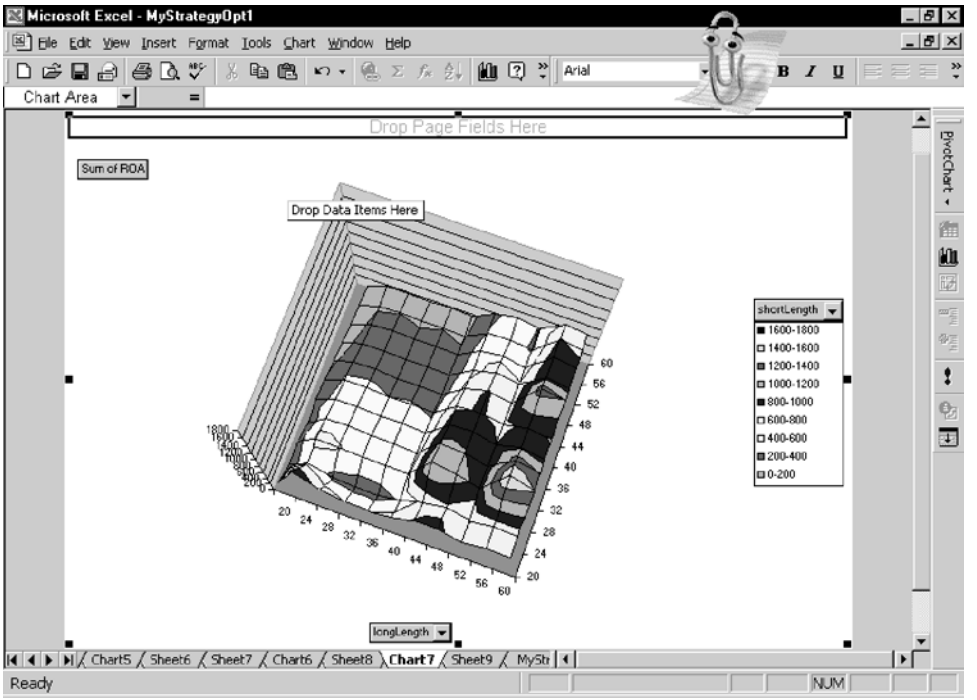

Figure 5.13 *(Continued)*

fall on elevated and level plateaus. You may not be able to see these parameters clearly by looking at the chart. You can change the chart view by going under the Chart menu and selecting 3-D view. We personally like to look down on the chart from an above perspective (an aerial view). This perspective allows you to see all of the peaks, valleys, and plateaus.

If you optimize more than two parameters, then it is impossible to visually determine parameter robustness. If you only optimize one parameter, you can do away with the PivotTable procedure and simply use the charting wizard in Excel. The more parameters you optimize, the higher the likelihood of curve fitting and overoptimization. If you have your heart set on optimizing more than two parameters, then do it in a stepwise fashion. Determine your two most important parameters and optimize and analyze for robustness. Once you select the most robust combination, optimize the other parameters and so on. Just be careful and don't fool yourself.

We all know the true test is the test of time. How a strategy performs after it is developed gives the most insight into the true validity of the strategy. Optimizing parameters over the entire history of a market does not allow for a "walk-forward test." A walk-forward test involves testing over data that was not used during the back test or strategy optimization. When searching for optimum parameters, we like to cut our historic database in half. The first half of the data is dedicated to back testing and optimization. The second half of the test is

reserved for forward testing our strategy and parameter selections. If the strategy or the parameters do not pass the walk-forward test period, then we go back to the drawing board. Walk-forward testing can actually be done on any out-of-sample data. Many people prefer to develop and optimize their strategies using recent history and then walk-forward test an earlier out-of-sample data. They believe that the most recent price history of a market gives more insight into the near future than does data that is several years old. We would have to agree that stocks and stock indices data look nothing like they did five years ago, so this may not be a bad idea for these types of markets. It doesn't matter which back testing/walk-forward combination you choose, as long as you choose one.

CONCLUSIONS

Understanding trading system performance and proper strategy optimization helps to determine a good strategy from a bad one. TradeStation offers a wealth of performance information on a particular market in the Strategy Performance report to do this. Unfortunately, it gives only a microscopic view of the robustness of our strategy. You can't develop a trading plan by looking at the performance of only one market; a macroscopic view is necessary before a strategy is to be given a stamp of approval. If you plan on allocating your trading capital across a handful of markets, you will be breaking the one true golden rule of trading: diversification. If you have any hopes of succeeding as a trader, you must diversify your capital across as many markets as possible. You can achieve diversifications through a portfolio of markets or trading strategies. Unfortunately, TradeStation is limited by its inability to analyze trading strategies at the portfolio level. Even though TradeStation is limited in this capacity, we can somewhat overcome this obstacle. If a trading strategy can be successful on a large and diverse portfolio of markets, then there is a good probability that it will be successful in the future. In other words, test, test, and retest. Throw the strategy against the wall as many times as necessary. Revel in failure, for every time you dismiss a losing strategy, you gain a victory in the battle for a winning trading strategy.

If a trading strategy is initially successful, we know that we can make it better through the process of optimization. Or, we make it less likely to be successful in the future. Optimization is a necessary evil. It is necessary so that we can come up with sound trading strategies (back testing is a form of optimization). At the same time, since we are human and enjoy tinkering with things, we can't help running 10,000 optimizations of five parameters in search of the Holy Grail. Hopefully, this chapter has introduced the knowledge to prevent the abuse of TradeStation and the vast database of historic data that is available at our fingertips. Chapter 6 will introduce turnkey trading strategies that rival any other trading strategies that can be purchased for $1000s.

6

Trading Strategies That Work (or The Big Damn Chapter on Trading Strategies)

Are you ready for some good trading strategy ideas and the programming behind them? Over the past twenty years, we have seen thousands of trading ideas, schemes, and systems. Unfortunately, most of them ended up in the trash can. Sometimes, however, we have scratched our heads with raised eyebrows and said, "Hmmm." The next strategies that we present are based on ideas that have proven successful time and time again. We present these systems as works in progress; as you will soon discover, trading system design has no end. No matter how long and arduously you work on a trading system, you will never be 100 percent satisfied with the results. We don't want you to be satisfied with the systems that we present. We want you to make them your own. Use the programs as templates and branch off in an entirely different direction.

The manner in which we present these trading systems will guide you through the steps necessary to translate a trading idea into software. We first start by describing the trading idea in general terms and our objectives. The tools that will be used to build our trading methodology are then defined. Pseudocode (half English and half computer speak) is then used to bridge the gap between system description and actual code. Finally, the exact EasyLanguage program is provided. Since testing several different markets over an extended time period is cumbersome with TradeStation, we present the overall and individual performance statistics of the system for you. Of course, you can also do this with your own TradeStation by building workspaces and applying each system.

We have tried to incorporate all of the necessary tools of system programming by creating rather sophisticated trading alogorithms. The complexity

level of each system grows as each one is presented. We hope the commentary that is included in the actual code will help clear up any confusion.

As the old saying goes, "There is more than one way to skin a cat." In most instances, there is more than one way to program a solution to a given problem; be it a complex, mathematical calculation or a trading system. You may and probably will come up with easier to understand and more efficient programs than we have presented here. We hope you do. Programming is a form of art, and like any other art, it is only limited by the creativity of the artist.

THE KING KELTNER TRADING STRATEGY

A moving average calculation is the main indicator used in the King Keltner trading strategy. A moving average is calculated by summing up *x* prior data points and then dividing the summation by *x*. Most times these calculations use a fixed number of data points. The more data points you have, the less of an impact a new data point has on the final averaged value. Longer moving average calculations try to determine longer-term trend movements. Conversely, shorter moving averages try to pinpoint shorter-term market swings. Chester Keltner presented this application of a moving average system in 1960. The system Keltner presented was built around a moving average of the high, low, and closing prices with a band or channel on each side of the market formed by a moving average of the high-low range. A buy signal occurs when the market penetrates the upper band and a sell signal when the market penetrates the lower. We have used the basic Keltner approach, but have added a few bells and whistles. We hope, as did Chester, that when the market makes an abrupt move away from its moving average, it is signaling a change in trend. In the King Keltner system, the penetration of the upper/lower bands signals this trend change. We will go with the flow and buy on strength and sell on weakness. We will get out with a win or a loss when the market retraces back to the moving average.

The major problem with channel break out systems is the failed breakout. Many times, the channels represent a point of market exhaustion instead of trend confirmation. Frequently, a market will spend itself by moving to the upper or lower bands and then immediately fall back and move in the opposite direction. This is our worst fear. However, since we realize the weakness of this type of system, we have programmed a liquidation stop at the moving average. Most trading methodologies will fail and some form of protection should be put into place when a trade is initiated. If most trading methodologies fail, then why put a trade on in the first place? The success to any form of trading is to cut losses short and let profits run. This basic tenet of trading falls under the realm of money management. Your trading system gets you into the trade and your money management scheme manages your position and eventually gets you out of the trade. In the King Keltner system, the direction of the moving average and the penetration of the bands are our entry technique, and the liquidation of our position at the moving average is our money management scheme. Our money management stop will either be a protective stop or a take profit stop. If we do capture a long trend, then the moving average should move in the same direction as our entry signal and with any luck capture a good portion of the move. Always remember it is the exit technique that determines the success of the entry technique. Since King Keltner is a long-term approach, short-term profits are not an objective. We will take them if they come our way, but with this type of system they would eventually become

counterproductive. This system will have fewer than 50 percent wins and that's all right. The few large trends that we do catch should more than cover the losses from the failed breakouts.

Most moving average-based systems are very simple to program and this one will not be an exception. We will need only two tools: (1) a moving average of the average of the high, low, and close prices, and (2) a moving average of the true ranges. You may not be familiar with the term *true range*. The range of a daily bar is simply calculated by subtracting the low price from the high price. An average of these ranges will give an estimate of future price ranges. The true range calculation extends the range of a bar to the previous day's close (true range = max(close of yesterday, high of today) – min(close of yesterday, low of today) thus, expanding the bar's range to include any gaps from the previous day's close. We feel true ranges give a slightly more accurate measure of market volatility. Since we are trying to capture a longer-term move, we will use 40 days in our average calculations.

King Keltner Pseudocode

```
movAvg = Average(((High + Low + Close)/3),40)
upBand = movAvg + Average(TrueRange,40)
dnBand = movAvg – Average(TrueRange,40)
liquidPoint = Average(((High + Low + Close)/3),40)

        A long position will be initiated when today's movAvg is greater than
           yesterday's and market action >= upBand
        A short position will be initiated when today's movAvg is less than
           yesterday's and market action <= dnBand
        A long position will be liquidated when today's market action
           <= liquidPoint
        A short position will be liquidated when today's market action
           >= liquidPoint
```

King Keltner Program

```
{King Keltner by George Pruitt–based on trading system presented by Chester
Keltner}
Inputs: avgLength(40), atrLength(40);
Vars: upBand(0),dnBand(0),liquidPoint(0),movAvgVal(0);
movAvgVal = Average((High + Low + Close),avgLength);
upBand = movAvgVal + AvgTrueRange(atrLength);
dnBand = movAvgVal – AvgTrueRange(atrLength);
if(movAvgVal > movAvgVal[1]) then Buy ("KKBuy") tomorrow at upBand stop;
if(movAvgVal < movAvgVal[1]) then Sell Short("KKSell") tomorrow at dnBand
   stop;
liquidPoint = movAvgVal;
```

```
If(MarketPosition = 1) then Sell tomorrow at liquidPoint stop;
If(MarketPosition = –1) then Buy To Cover tomorrow at liquidPoint stop;
```

The King Keltner program demonstrates how to:

- Invoke the Average and Average True Range functions.
- Buy/Sell on the next bar at a stop level.
- Liquidate a position on the next bar at a stop level.
- Incorporate inputs for user interface and future optimizations.

King Keltner trading performance is summarized in Table 6.1

A visual example of how this system enters and exits trades is shown in Figure 6.1.

Table 6.1
King Keltner Performance

System Name: King Keltner Commission/Slippage = $75
Tested 1982 – 3/19/2002

Markets	Total Net Profit	Max. DrawDown	# of Trades	% Wins	Max. Cons. Losers
British Pound	$ 48,056.25	$ (51,962.50)	239	30.13%	25
Crude Oil	$ 36,152.50	$ (17,682.50)	184	32.07%	16
Corn	$ (612.50)	$ (10,681.25)	251	22.71%	14
Copper	$ 5,180.00	$ (12,182.50)	149	33.56%	10
Cotton	$ 30,387.50	$ (26,997.50)	241	24.48%	15
Deutsch Mark	$ 57,962.50	$ (11,575.00)	208	33.17%	10
Euro Currency	$ 2,612.50	$ (9,425.00)	36	38.89%	5
Euro Dollar	$ 37,392.50	$ (6,130.00)	204	30.88%	21
Heating Oil	$ 10,673.68	$ (25,697.71)	240	27.50%	12
Japanese Yen	$ 114,175.00	$ (30,162.50)	215	31.16%	12
Live Cattle	$ (3,036.50)	$ (21,925.50)	243	24.28%	24
Natural Gas	$ 100,577.50	$ (14,157.50)	119	37.82%	7
Soybeans	$ (15,193.75)	$ (34,818.75)	251	27.49%	15
Swiss Franc	$ 56,962.50	$ (14,837.50)	220	32.27%	8
Treasury Note	$ 61,850.00	$ (11,053.13)	209	33.01%	10
U.S. Bonds	$ 66,275.00	$ (15,543.75)	215	28.84%	9
Wheat	$ (16,112.50)	$ (19,906.25)	254	22.83%	14
Total	$ 593,302.18		3478		

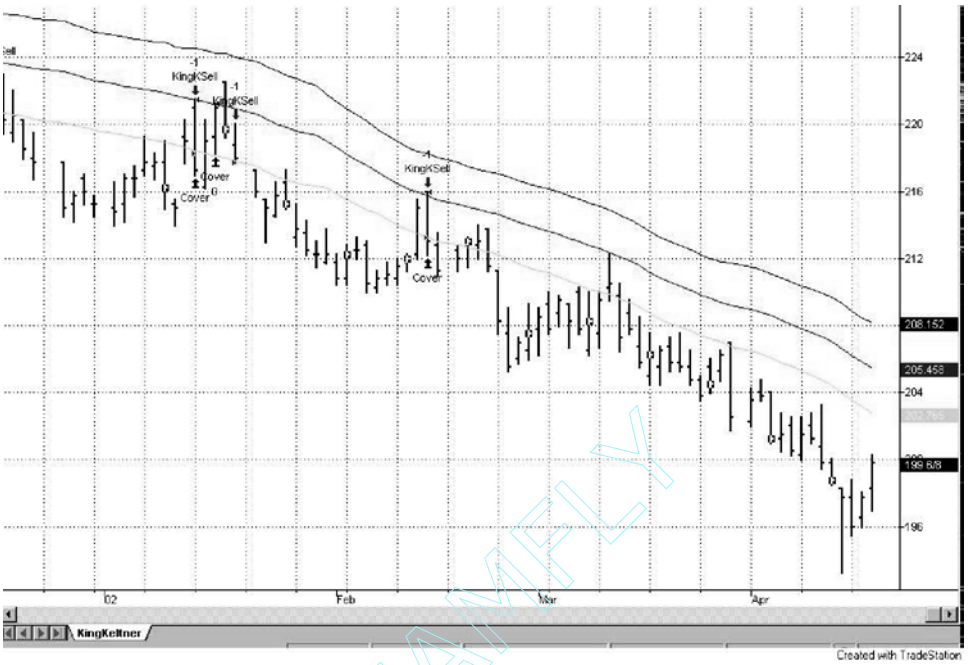

Figure 6.1 King Keltner Trades

King Keltner Summary

Overall trading performance was extremely positive. The system did well in the majority of the test markets, which is a testament to its robustness. Remember there are only two parameters, which are the same for all markets. Could this system be improved by optimizing the parameters on an individual market basis? We like the idea of the same parameter set, but others in the industry would argue this point with us. Their argument would be based on the belief that markets from different sectors (e.g., Japanese Yen and live cattle) have different underlying fundamentals and, therefore, do not demonstrate similar market movements. Changing a parameter to reflect the differences between different markets is not just acceptable but it is an absolute necessity. We don't totally agree with this argument, but we could be talked into different parameters for different sectors. All of the currencies would have one set of parameters, and all of the meats would have one and so on. We would emphatically disagree with the idea of having a different parameter for the Japanese Yen and the Swiss Franc; these two markets have similar fundamentals and market movements. King Keltner could be the foundation for an entire portfolio-based trading platform. All that is needed is an algorithm for bet size (the number of contracts that is put on with each trade). In other words, you would need a money management overlay.

THE BOLLINGER BANDIT TRADING STRATEGY

Standard deviation is a number that indicates how much on average each of the values in the distribution deviates from the mean (or center) of the distribution. Bollinger Bands, created by John Bollinger in the 1960s, is an indicator that uses this statistical measure to determine support and resistance levels. This indicator consists of three lines and is very simple to derive; the middle line is a simple moving average of the underlying price data and the two outside bands are equal to the moving average plus or minus one standard deviation. Based on theory, two standard deviations equates to a 95 percent confidence level. In other words, 95 percent of the time the values used in our sampling fell within two standard deviations of the average. Initially, Bollinger Bands were used to determine the boundaries of market movements. If a market moved to the upper band or lower band, then there was a good chance that the market would move back to its average. We have carried out numerous tests on this hypothesis and seemed to always come back with failure. Instead of using the upper band as a resistance point, we discovered, as others have, that it worked much better as a breakout indicator. The same goes for the lower band. The Bollinger Bandit uses one standard deviation above the 50-day moving average as a potential long entry and one standard deviation below the 50-day moving average as a potential short entry. This system is a first cousin of King Keltner. They are similar in that they are longer-term channel breakout systems. However, this is where the similarities end. Instead of simply liquidating a position when the market moved back to the moving average, we concocted a little twist to this exit technique. From observing the trades on the King Keltner, we discovered that we gave back a good portion of the larger profits waiting to exit the market at the moving average. So, for the Bollinger Bandit, we incorporated a more aggressive trailing stop mechanism. When a position is initiated, the protective stop is set at the 50-day moving average. Every day that we are in a position, we decrement the number of days for our moving average calculation by one. The longer that we are in a trade, the easier it is to exit the market with a profit. We keep decrementing the number of days in our moving average calculation until we reach ten. From that point on, we do not decrement. There is one more element to our exit technique: the moving average must be below the upper band if we are long and above the lower band if we are short. We added this element to prevent the system from going back into the same trade that we just liquidated. If we hadn't used this additional condition and we were long and the moving average was above the upper band, the long entry criteria would still be set up and a long trade would be initiated.

Previously, we stated that the upper band and lower band were potential buy/sell entries. Potential is the key word. One more test must be passed before we initiate a position; the close of today must be greater than the close of 30 days ago for a long position and the close of today must be less than the

close of 30 days ago for a short position. This additional requirement is a trend filter. We only want to go long in an uptrend and short in a downtrend.

The Bollinger Bandit requires four tools: (1) Bollinger Bands, (2) a moving average of closing prices, (3) a rate of change calculation, and (4) a counter. This system is longer term in nature, so we will use 50 days in our calculations.

Bollinger Bandit Pseudocode

```
LiqDay is initially set to 50
upBand = Average(Close,50) + StdDev(Close,50) *1.25
dnBand = Average(Close,50) - StdDev(Close,50) *1.25
rocCalc = Close of today - Close of thirty days ago

        Set liqLength to 50
        If rocCalc is positive, a long position will be initiated when
           today's market action >= upBand
        If rocCalc is negative, a short position will be initiated when
           today's market action <= dnBand
        liqPoint = Average(Close, 50)
        If liqPoint is above the upBand, we will liquidate a long position if
           today's market action <= liqPoint
        If liqPoint is below the dnBand, we will liquidate a short position
           if today's market action >= liqPoint
        If we are not stopped out today, then liqLength = liqLength - 1
        If we are stopped out today, then reset liqLength to fifty
```

Bollinger Bandit Program

```
{Bollinger Bandit by George Pruitt—program uses Bollinger Bands and Rate of
change to determine entry points. A trailing stop that is proportional with
the amount of time a trade is on is used as the exit technique.}

Inputs: bollingerLengths(50),liqLength(50),rocCalcLength(30);
Vars: upBand(0),dnBand(0),liqDays(50),rocCalc(0);
upBand = BollingerBand(Close,bollingerLengths,1.25);
dnBand = BollingerBand(Close,bollingerLengths,-1.25);

rocCalc = Close - Close[rocCalcLength-1]; {remember to subtract 1}
if(MarketPosition <> 1 and rocCalc > 0) then Buy("BanditBuy")tomorrow upBand
   stop;
if(MarketPosition <>-1 and rocCalc < 0) then SellShort("BanditSell") tomorrow
   dnBand stop;

if(MarketPosition = 0) then liqDays = liqLength;
if(MarketPosition <> 0) then
begin
       liqDays = liqDays - 1;
       liqDays = MaxList(liqDays,10);
```

```
end;
if(MarketPosition = 1 and Average(Close,liqDays) < upBand) then
       Sell("Long Liq") tomorrow Average(Close,liqDays) stop;
if(MarketPosition = -1 and Average(Close,liqDays) > dnBand) then
       BuyToCover("Short Liq") tomorrow Average(Close,liqDays) stop;
```

The Bollinger Bandit program demonstrates how to:

- Invoke the Bollinger Band function. This function call is less than intuitive and must be passed three parameters: (1) price series, (2) number of elements in the sample used in the calculation for the standard deviation, and (3) number of deviations above/below moving average. You must use a negative sign in the last parameter to get the band to fall under the moving average.
- Invoke the MaxList function. This function returns the largest value in a list.
- Do a simple rate of change calculation.
- Create and manage a counter variable, liqLength.

Bollinger Bandit trading performance is summarized in Table 6.2.

Table 6.2
Bollinger Bandit Performance

System Name: Bollinger Bandit Commission/Slippage = $75
Tested 1982 – 3/19/2002

Markets	Total Net Profit	Max. DrawDown	# of Trades	% Wins	Max. Cons. Losers
British Pound	$ 38,750.00	$ (43,612.50)	194	33.51%	20
Crude Oil	$ 47,242.50	$ (17,522.50)	170	41.76%	8
Corn	$ (5,112.50)	$ (12,937.50)	213	29.58%	13
Copper	$ 2,300.00	$ (9,587.50)	138	36.23%	12
Cotton	$ 26,695.00	$ (12,437.50)	220	32.73%	8
Deutsch Mark	$ 51,075.00	$ (13,812.50)	186	41.40%	6
Euro Currency	$ 8,737.50	$ (9,012.50)	29	44.83%	7
Euro Dollar	$ 31,927.50	$ (6,622.50)	196	35.71%	19
Heating Oil	$ 16,883.14	$ (18,378.89)	201	38.81%	10
Japanese Yen	$ 121,937.50	$ (21,462.50)	180	37.22%	8
Live Cattle	$ (16,867.50)	$ (25,411.50)	224	26.79%	18
Natural Gas	$ 85,897.50	$ (21,737.50)	113	44.25%	6
Soybeans	$ (15,925.00)	$ (40,862.50)	215	31.16%	15
Swiss Franc	$ 76,312.50	$ (9,987.50)	188	40.96%	5
Treasury Note	$ 39,625.00	$ (11,487.50)	202	38.12%	9
U.S. Bonds	$ 48,381.25	$ (15,343.75)	204	36.27%	6
Wheat	$ (20,037.50)	$ (21,931.25)	219	29.68%	11
Total	$ 537,821.89		3092		

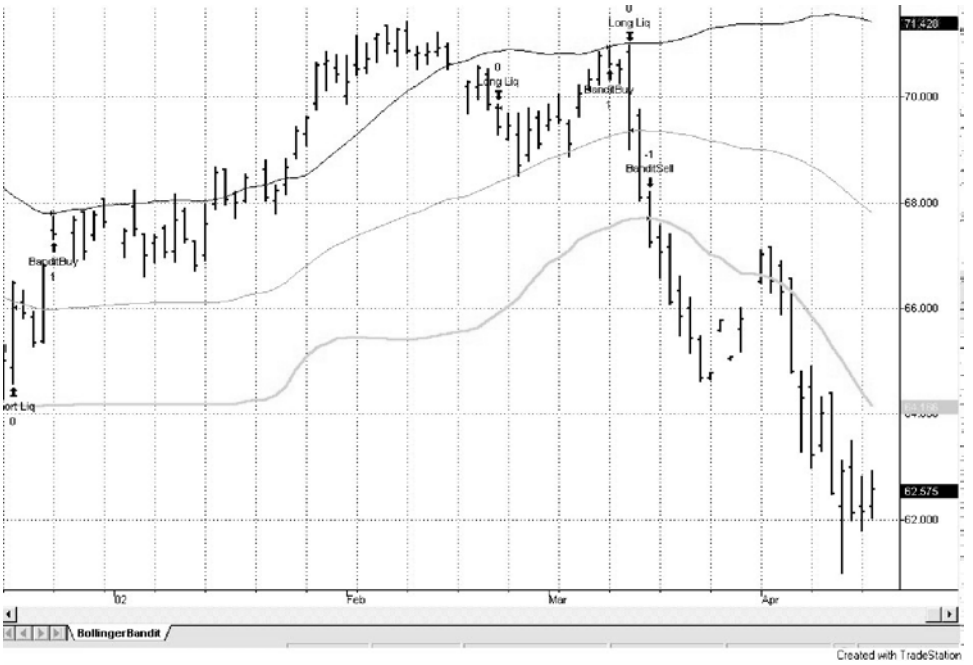

Figure 6.2 Bollinger Bandit Trades

A visual example of how this system enters and exits trades is shown in Figure 6.2.

Bollinger Bandit Summary

Overall trading performance was positive. You can see the similarities between the Bollinger and Keltner-based systems. The same markets that made good money in one system made good money in the other. These systems would not work well together due to their high level of correlation. This system did exceptionally well in the Japanese Yen and Natural Gas. Through further investigation, we discovered that our trailing stop mechanism only marginally increased profit and decreased draw down. Nonetheless, the concept probably adds a higher comfort level when a trade is initiated. We know that our risk should diminish the farther we get into a trade. This is due to the fact that a shorter-term moving average follows closer to the actual market than a longer-term average.

THE THERMOSTAT TRADING STRATEGY

We actually traded a strategy very similar to Thermostat. We named this system based on its ability to switch gears and trade in the two modes of the market, congestion and trend. This system sprung from our observations on the success of particular systems on particular market sectors. We thought that one system that also possessed a dual nature could be created to capitalize on the market's two modes. We created a function to help determine the market's mode. Based on the output of this function, the Thermostat switches from a trend-following mode to a short-term swing mode.

The trend-following mode uses a trend following mechanism similar to the one found in the Bollinger Bandit. The short-term swing system is an open range breakout that incorporates pattern recognition. The function that causes the system to switch its approach is the same function that we described in Chapter 4, the ChoppyMarketIndex. If you have forgotten this function, here is a brief review. This function compares the distance the market wanders and the actual distance the market traveled (Abs(Close – Close[29])/(Highest(High,30) – Lowest(Low,30))*100). The function generates values from 0 to 100. The larger the value, the less congested the current market is.

If the ChoppyMarketIndex function returns a value of less than 20, then the system goes into the short-term swing mode. Basically, the market is demonstrating a swinging motion and the system tries to catch the swings and pull out a small profit. Thermostat tries to accomplish this feat by buying/selling on small market impulses. If the impulse is large enough, then the system jumps on and tries to ride the market out for the next few days. Through in-depth analysis of short-term swings, we have discovered that there exist certain days when it is better to buy than sell and vice versa. These days can be determined by a simple visual pattern. If today's closing price is greater than the average of yesterday's high, low, and close (also known as the key of the day), then we feel tomorrow's market action will probably be bearish. However, if today's closing price is less than the average of yesterday's high, low, and close, then today's market will probably behave in a bullish manner. We classify these days as sell easier and buy easier days, respectively. We know that we can't predict the market, and this is the reason we use the term, *easier*. Even though today may be a buy easier day, we can still sell and vice versa. We just make it easier to buy on buy easier days and sell on sell easier days. A position is triggered when the market moves a certain range from the open price.

If today is a buy easier day then:

```
Initiate a long position at the open price + 50% of the ten-day average true
  range.
Initiate a short position at the open price – 100% of the ten-day average
  true range.
```

If today is a sell easier day then:

```
Initiate a short position at the open price – 50% of the ten-day average true
   range.
Initiate a long position at the open price + 100% of the ten-day average true
   range.
```

Sometimes the market will have a false impulse in the opposite direction of the short-term swing. These types of impulses can whipsaw you in and out of the market and only make money for your broker. We try to prevent this by comparing our calculated buy stop with the three-day moving average of the low prices. If our calculated buy stop is less than the three-day moving average, we then move the buy stop up to the average calculation. If our sell stop is greater than the three-day moving average of the high prices, we then move a sell stop down to the three-day average. The system is in the market 100 percent of the time, when we are in choppy mode. Our short-term strategy of Thermostat is: If we have a move, we will be ready for it. It seems rather complicated, but once we get it programmed we can forget about the complexity.

If the ChoppyMarketIndex function returns a value greater than or equal to 20, then the system goes into the long-term trend-following mode. Our function has basically informed us that the market is moving in a general direction, without a bunch of noise. One of the best trend following approaches that we have seen is the same approach that we used in the Bollinger Bandit. A long position is initiated when the market breaks through the upper Bollinger Band and initiates a short position when the market breaks through the lower Bollinger Band. In the case of the trend-following component of Thermostat, we used two standard deviations in our calculation, instead of the 1.25 we used in the Bollinger Bandit. If we have a long position, then we liquidate if the price moves back to the moving average and vice versa. We use the same 50-day moving average as we did before.

Many times, you will have a position on when the market switches modes. If we switch from trending to congestion, we simply use the short-term entry method to get out of our trend mode position. However, if the market switches from congestion to trending and we have a position on, we then use a three-average true-range protective stop. This type of stop is utilized because the 50-day moving average exit that we used in trend mode is not congruent with our short-term entry technique. When designing trading systems, your entry and exit techniques must have similar time horizons. You wouldn't use two-day low trailing stop on a long position that was initiated by the crossing of a 75-day moving average. If we are long, we calculate the average true range for the past ten days and multiply this calculation by three and subtract the result from our entry price. If we are short, we again calculate the average true range for the past ten days and multiply this calculation by three but then add the result to

our entry price. Once we exit the positions that were initiated in choppy mode, we begin using the trend-following system to initiate any new signals.

Thermostat Pseudocode

Determine current market mode by using the ChoppyMarketIndex function. If the ChoppyMarketIndex function returns a value of less than 20, then use the short-term swing approach.

```
atr10 = AverageTrueRange(10)
keyOfDay = (High + Low + Close)/3
buyEasierDay = 0
sellEasierDay = 0
if(Close > keyOfDay) then sellEasierDay = 1
if(Close<=keyOfDay) then buyEasierDay = 1
avg3Hi = Average(High,3)
avg3Lo = Average(Low,3)
if(buyEasierDay = 1) then
       longEntryPoint = Open + atr10 * 0.5
       shortEntryPoint = Open - atr10 * 0.75
if(sellEasierDay = 1) then
       longEntryPoint = Open + atr10 * 0.75
       shortEntryPoint = Open - atr10 * 0.5
longEntryPoint = MaxList(longEntryPoint,avg3Lo)
shortEntryPoint = MinList(shortEntryPoint,avg3Hi)
Initiate a long position of today's market action >= longEntryPoint
Initiate a short position of today's market action <= shortEntryPoint
```

If the ChoppyMarketIndex function returns a value greater than or equal to 20, then use the long-term trend following approach.

If you have a short position that was initiated by the short-term swing approach then

```
       shortLiqPoint = entryPrice + 3 * atr10
       Liquidate short position if today's market action
           >= shortLiqPoint
```

If you have a long position that was initiated by the short-term swing approach then

```
       longLiqPoint = entryPrice - 3 * atr10
       Liquidate long position if today's market action
          <= longLiqPoint
upBand = Average(Close,50) + StdDev(Close,50) * 2.00
dnBand = Average(Close,50) - StdDev(Close,50) * 2.00
avg50 = Average(Close,50)
Initiate a long position if today's market action >= upBand
```

```
       Initiate a short position if today's market action <= dnBand
       Liquidate long position if today's market action <= avg50
       Liquidate short position if today's market action >= avg50
```

Thermostat Program

```
{Thermostat by George Pruitt
Two systems in one. If the ChoppyMarketIndex is less than 20 then we are in a
swing mode. If it is greater than or equal to 20 then we are in a trend mode.
Swing system is an open range breakout incorporating a buy easier/sell easier
concept. The trend following system is based on bollinger bands and is
similar to the Bollinger Bandit program.}

Inputs: bollingerLengths(50),trendLiqLength(50),numStdDevs(2),
swingPrcnt1(0.50),swingPrcnt2(0.75),atrLength(10),
swingTrendSwitch(20);
Vars:cmiVal(0),buyEasierDay(0),sellEasierDay(0),trendLokBuy(0),
trendLokSell(0),keyOfDay(0),swingBuyPt(0),swingSellPt(0),
trendBuyPt(0),trendSellPt(0),swingProtStop(0);

cmiVal = ChoppyMarketIndex(30);
buyEasierDay = 0;
sellEasierDay = 0;

trendLokBuy = Average(Low,3);
trendLokSell= Average(High,3);

keyOfDay = (High + Low + Close)/3;
if(Close > keyOfDay) then sellEasierDay = 1;
if(Close <= keyOfDay) then buyEasierDay = 1;

if(buyEasierDay = 1) then
begin
       swingBuyPt = Open of tomorrow + swingPrcnt1*AvgTrueRange(atrLength);
       swingSellPt = Open of tomorrow - swingPrcnt2*AvgTrueRange(atrLength);
end;
if(sellEasierDay = 1) then
begin
       swingBuyPt = Open of tomorrow + swingPrcnt2*AvgTrueRange(atrLength);
       swingSellPt = Open of tomorrow - swingPrcnt1*AvgTrueRange(atrLength);
end;

swingBuyPt = MaxList(swingBuyPt,trendLokBuy);
swingSellPt = MinList(swingSellPt,trendLokSell);

trendBuyPt = BollingerBand(Close,bollingerLengths,numStdDevs);
trendSellPt = BollingerBand(Close,bollingerLengths,- numStdDevs);
```

```
if(cmiVal < swingTrendSwitch)then
begin
       if (MarketPosition <> 1) then Buy("SwingBuy") next bar at swingBuyPt
           stop;
       if(MarketPosition <> -1) then SellShort("SwingSell") next bar at
          swingSellPt stop;
end
else
begin
       swingProtStop = 3*AvgTrueRange(atrLength);
       Buy("TrendBuy") next bar at trendBuyPt stop;
       SellShort("TrendSell") next bar at trendSellPt stop;
       Sell from Entry("TrendBuy") next bar at Average(Close,trendLiqLength)
          stop;
       BuyToCover from Entry("TrendSell") next bar at
          Average(Close,trendLiqLength) stop;
       Sell from Entry("SwingBuy") next bar at EntryPrice - swingProtStop
          stop;
       BuyToCover from Entry("SwingSell") next bar at EntryPrice +
          swingProtStop stop;
end;
```

The Thermostat program demonstrates how to:

- Invoke the ChoppyMarketIndex function. This function simply needs the number of bars in the calculation to be passed to it.
- Properly use an if-then-else control structure.
- Tie an exit strategy to a specific entry strategy using the *from entry* keywords.
- Calculate the key of the day. Also known as the day trader's pivot point.
- Use the next day's opening price in the calculation of our buy/sell orders.

Thermostat trading performance is summarized in Table 6.3.

A visual example of how this system enters and exits trades is shown in Figure 6.3.

Thermostat Summary

This program performed admirably in most of the markets. The synthesis of the two different approaches seemed like the way to go in the interest-bearing markets, Treasury bonds, and Treasury notes. Again, we were able to demonstrate an approach with one set of parameters for all markets. If there is only one thing you take away with you from this book, it should be the knowledge

Table 6.3
Thermostat Performance

System Name: Thermostat Commission/Slippage = $75
Tested 1982–3/19/2002

Markets	Total Net Profit	Max. DrawDown	# of Trades	% Wins	Max. Cons. Losers
British Pound	$ 24,518.75	$ (59,500.00)	260	35.00%	14
Crude Oil	$ 37,172.50	$ (15,927.50)	242	34.71%	10
Corn	$ 15,987.50	$ (5,900.00)	243	37.04%	9
Copper	$ (6,087.50)	$ (18,325.00)	205	31.71%	9
Cotton	$ 2,490.00	$ (22,707.50)	311	32.48%	11
Deutsch Mark	$ 60,362.50	$ (20,200.00)	226	34.96%	8
Euro Currency	$ 24,687.50	$ (12,600.00)	33	54.55%	5
Euro Dollar	$ 37,255.00	$ (9,320.00)	235	33.62%	14
Heating Oil	$ 13,448.13	$ (34,497.29)	279	32.62%	15
Japanese Yen	$ 121,250.00	$ (23,400.00)	189	36.51%	7
Live Cattle	$ 3,572.50	$ (23,887.50)	290	32.76%	15
Natural Gas	$ 83,272.50	$ (18,272.50)	152	36.84%	7
Soybeans	$ 35,331.25	$ (61,325.00)	319	28.84%	15
Swiss Franc	$ 124,887.50	$ (13,250.00)	206	44.66%	7
Treasury Note	$ 101,075.00	$ (11,621.88)	231	40.26%	11
U.S. Bonds	$ 103,081.25	$ (13,562.50)	268	41.79%	9
Wheat	$ (21,231.25)	$ (23,218.75)	331	28.10%	14
Total	$ 761,073.13		4020		

that market principles are generally universal. Thermostat falls in the category of intermediate term trend follower; it generates around 15 to 20 trades per year. Due to its ability to trade a shorter time horizon, Thermostat would probably be a good candidate to trade in concert with a longer-term approach. If you look at the trade-by-trade analysis, you probably won't see many trades labeled as TrendBuy or TrendSell. This may lead you to believe that the system was mostly in the choppy trading mode. You would be wrong. In fact, the system was considerably more in the trending mode (as defined as a ChoppyMarketIndex reading above 20). Many times after a trade is initiated in the choppy or swing mode, the overall market mode switches to trending and stays in that mode for an extended period of time. If the market changes mode and doesn't trend in the direction of the trade, it eventually is stopped out. If the market does trend in the direction of the trade, then the position is held until

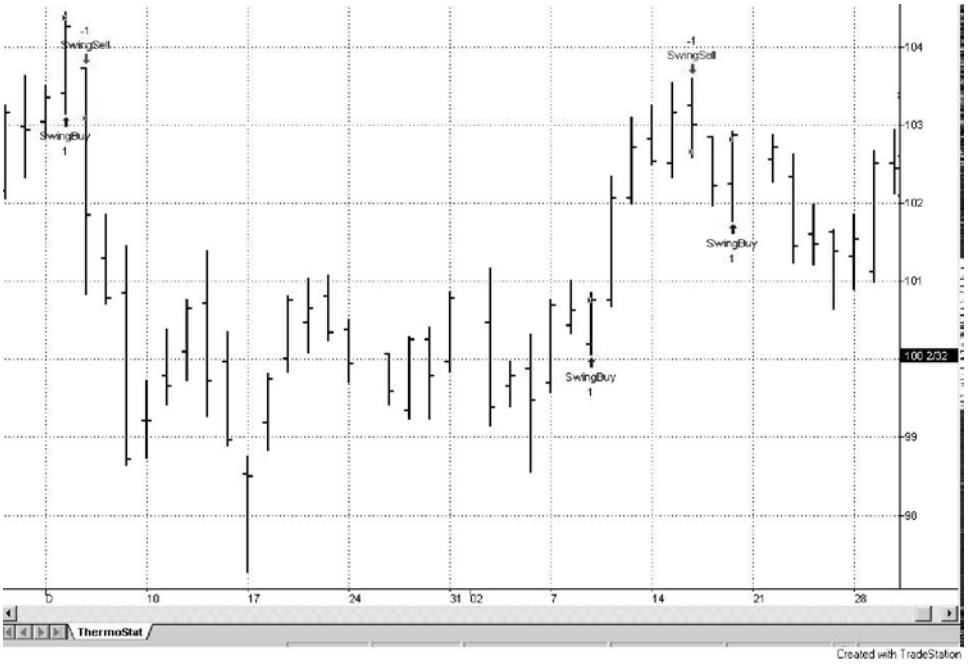

Figure 6.3 Thermostat Trades

the market either ends the trend or changes mode. Either way, the trade actually turns out to be a trend trade instead of a choppy or swing trade. If there is a large profit, then the Swing Buy/Sell gets the credit. In other words, the system performance is generated by both systems, even though it doesn't look like it.

THE DYNAMIC BREAK OUT II STRATEGY

George Pruitt for *Futures Magazine* designed the original Dynamic Break Out system in 1996. This version has done well since it was released for public consumption in 1996. This version will be included in Appendix B. The newer version of the Dynamic Break Out is just like the original, except we have incorporated an additional adaptive filter.

The key to the Dynamic Break Out II system is its ability to adapt its parameters to current market conditions. This system is based on the tried-and-tested Donchian channel system. Remember how the Donchian system works; buy when the high of the day penetrates the highest high price of *x* bars back, and sell when the low of the day penetrates the lowest low of *x* bars back. If you optimize the number of bars to determine your best entry and exit levels, you will discover that different markets work better with different parameters. You will also discover that a particular market goes through different cycles and works better with different parameters through time. For example, the Japanese Yen may have performed better with a look back of 40 days in the 1980s, but now works better with a look back of 20 days. That is the major problem with using a static parameter for all markets. The Dynamic Break Out II system allows the number of look back days to change with the current market. Instead of using a static parameter, this system changes the parameters based on an aspect of the current market.

Before you can use an adaptive parameter, you must come up with a function or adaptive engine that automatically changes the value of the once static parameter. The input of this adaptive engine should be some form of market statistic. In the case of the Dynamic Break Out II, we used market volatility. When market volatility expands, so does the number of look back days in our break out calculation. Increased market volatility usually equates to market indecisiveness. By increasing the number of look back days when market volatility increases, we make it more difficult for the system to initiate a trade. When market volatility decreases, we reduce the number of look back days. Low market volatility equates to a trending market. By decreasing the number of look back days, we encourage the system to initiate a trade. This helps the Dynamic Break Out II to lock into long-term profits and be on the look out for a change in the long-term trend. We used market volatility to fuel our adaptive engine, but you could use any market characteristic. We can visualize an engine that uses a market's overbought/oversold state. If we had a long position in a market, and it became overbought, we could use an overbought/oversold indicator to adapt the parameter that determines the sell point.

Once an adaptive engine is dreamed up and it is pumping out values, you must maintain the values in an acceptable range. The Dynamic Break Out II system will not let the look back days go above 60 or below 20. Through optimization, we discovered that look back lengths that fell beyond these bound-

aries did not generate acceptable expectations. An adaptive engine that generates useless values is useless in itself.

The Dynamic Break Out II initially looks back 20 days to determine its buy and sell levels. So when you start trading this system, your first buy point is the highest high of the past 20 days and your sell point is the lowest low of the past 20 days. At the end of each day, you measure the current market volatility by calculating the standard deviation of the past 30 day's closing prices. Market volatility can be measured using different calculations: average range, average true range, standard deviation of change in closing prices, and others. Once we determine today's market volatility, we compare it with yesterday's. If the volatility increases, then the number of look back days also increases. We change the number of look back days to the exact amount of the change in market volatility; if volatility increases by ten percent, then so does the number of look back days and vice versa.

The original Dynamic Break Out made its buying and selling decisions solely based on the highest high and lowest low values that were generated by our volatility-based adaptive engine. Once a position was initiated, a simple, yet effective, $1500 money management stop was put into place. The newer version uses the same entry technique in concert with an adaptive Bollinger Band. The length of the Bollinger Band calculation is the same number of look back days that is generated by the adaptive engine. The close of yesterday must be above the upper band and today's high must be greater than or equal to the highest high of *x* bars back before a long position can be initiated (*x* bars back is equal to our adaptive look back days value). Yesterday's close must be below the lower band and today's low must be less than or equal to the lowest low of *x* bars back before a short position can be taken. Instead of the simple money management stop, we incorporated a dynamic trailing stop. As we have discussed, the number of look back days changes on a daily basis. The adaptive engine decides the amount of change. The liquidation point of an existing trade is determined by calculating a simple moving average of closing prices for the past look back days. The sell liquidation would be just the opposite of the buy liquidation.

Dynamic Break Out II Pseudocode

```
If BarNumber = 1 then lookBackDays = 20
Else do the following
       Today's market volatility = StdDev(Close,30)
       Yesterday's market volatility = StdDev(Close[1],30)
       deltaVolatility = (today's volatility - yesterday's
          volatility)/today's volatility
       lookBackDays = (1 + deltaVolatility) * lookBackDays
       lookBackDays = MinList(lookBackDays,60)
       lookBackDays = MaxList(lookBackDays,20)
```

```
upBand = Average(Close,lookBackDays) + StdDev(Close,lookBackDays) *2.00
dnBand = Average(Close,lookBackDays) - StdDev(Close,lookBackDays) *2.00
buyPoint = Highest(High,lookBackDays)
sellPoint = Lowest(Low,lookBackDays)
longLiqPoint = Average(Close,lookBackDays)
shortLiqPoint = Average(Close,lookBackDays)
If Close of yesterday > upBand) then initiate a long position if today's
   market action >= buyPoint
If (Close of yesterday < dnBand) then initiate a short position if today's
   market action <= sellPoint
Liquidate long position if today's market action <= longLiqPoint
Liquidate short position if today's market action >= shortLiqPoint
```

Dynamic Break Out II Program

```
{Dynamic Break Out II by George Pruitt
This system is an extension of the original Dynamic Break Out system written
by George for Futures Magazine in 1996. In addition to the channel break out
methodology, DBS II incorporates Bollinger Bands to determine trade entry.}

Inputs: ceilingAmt(60),floorAmt(20),bolBandTrig(2.00);
Vars: lookBackDays(20),todayVolatility(0),yesterDayVolatility(0),
   deltaVolatility(0);
Vars: buyPoint(0),sellPoint(0),longLiqPoint(0),shortLiqPoint(0),upBand(0),
   dnBand(0);

todayVolatility = StandardDev(Close,30,1);
yesterDayVolatility = StandardDev(Close[1],30,1); {See how I offset the
                                                    function call to get
                                                    yesterday's value}
deltaVolatility = (todayVolatility - yesterDayVolatility)/todayVolatility;
lookBackDays = lookBackDays * (1 + deltaVolatility);
lookBackDays = Round(lookBackDays,0);
lookBackDays = MinList(lookBackDays,ceilingAmt); {Keep adaptive engine within
                                                  bounds}
lookBackDays = MaxList(lookBackDays,floorAmt);
upBand = BollingerBand(Close,lookBackDays,+bolBandTrig);
dnBand = BollingerBand(Close,lookBackDays,-bolBandTrig);

buyPoint = Highest(High,lookBackDays);
sellPoint = Lowest(Low,lookBackDays);

longLiqPoint = Average(Close,lookBackDays);
shortLiqPoint = Average(Close,lookBackDays);

if(Close > upBand) then Buy("DBS-2 Buy") tomorrow at buyPoint stop;
if(Close < dnBand) then SellShort("DBS-2 Sell") tomorrow at sellPoint stop;
```

```
if(MarketPosition = 1) then Sell("LongLiq") tomorrow at longLiqPoint stop;
if(MarketPosition = -1) then BuyToCover("ShortLiq") tomorrow at shortLiqPoint
   stop;
```

The Dynamic Break Out II program demonstrates how to:

- Measure market volatility by using the standard deviation of closing prices.
- Create a dynamic parameter using an adaptive engine

Dynamic Break Out II trading performance is summarized in Table 6.4.

A visual example of how this system enters and exits trades is shown in Figure 6.4.

Table 6.4
Dynamic Break Out II Performance

System Name: Dynamic Breakout Commission/Slippage = $75
Tested 1982–3/19/2002

Markets	Total Net Profit	Max. DrawDown	# of Trades	% Wins	Max. Cons. Losers
British Pound	$ 38,750.00	$ (43,612.50)	194	33.51%	20
Crude Oil	$ 21,237.50	$ (15,312.50)	109	35.78%	10
Corn	$ 3,050.00	$ (7,887.50)	120	34.17%	13
Copper	$ (25,175.00)	$ (25,862.50)	86	30.23%	7
Cotton	$ 25,555.00	$ (12,427.50)	112	33.93%	7
Deutsch Mark	$ 49,087.50	$ (7,837.50)	103	46.60%	5
Euro Currency	$ (7,062.50)	$ (10,950.00)	14	28.57%	4
Euro Dollar	$ 16,885.00	$ (5,025.00)	110	38.18%	8
Heating Oil	$ 30,728.10	$ (12,443.09)	113	39.82%	13
Japanese Yen	$ 118,200.00	$ (10,087.50)	98	51.02%	4
Live Cattle	$ (17,396.50)	$ (21,119.50)	125	25.60%	12
Natural Gas	$ 51,557.50	$ (14,902.50)	65	40.00%	7
Soybeans	$ (9,681.25)	$ (28,237.50)	128	33.59%	10
Swiss Franc	$ 57,337.50	$ (13,850.00)	106	47.17%	4
Treasury Note	$ 47,168.75	$ (6,646.88)	106	36.79%	7
U.S. Bonds	$ 67,093.75	$ (16,006.25)	107	40.19%	6
Wheat	$ (14,831.25)	$ (17,256.25)	124	31.45%	9
Total	$ 452,504.10		1820		

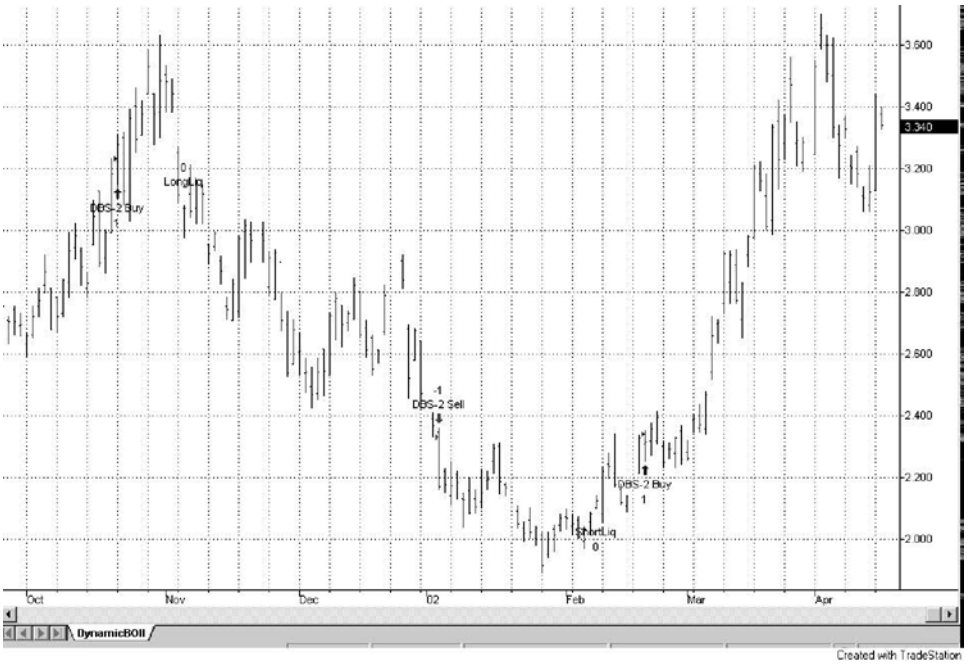

Figure 6.4 Dynamic Break Out II Trades

Dynamic Break Out II Summary

Yet again, another successful long-term trading approach. We guess we let the cat out of the bag . . .and what an ugly cat it is. The majority of successful trading systems are of the long-term trend following variety. Almost all traders realize this fact, but it doesn't stop them from searching out a shorter-term approach. See, the trend-following systems require diversification, which requires hefty capitalization. Also, trend-following systems can have substantial draw downs and go for years without making any money. The typical trader cannot persevere through these bad attributes, even though they know they will probably be rewarded in the long run.

Even this dynamic approach couldn't capture a profit in the soybean market. The continual failure of trend-following systems in the grain markets begs the question, "Why don't these systems work in the soybean or grain markets?" These markets move in a cyclical fashion due to the seasonality aspect of their underlying fundamentals. If we know ahead of time that these markets have this cyclical nature, then why can't we capture their movements? Cycles are very difficult to calculate and determine and, therefore, are usually overlooked. The two most predominant methods for finding cycles are trigonometric curve fitting and Fourier (spectral) analysis. The mathematics behind these two methods is relatively complex and detailed. We personally have

never seen a pure mathematically-based, cycle-finding trading system outperform the typical trend follower. If you do have any interest in this area, we refer you to John Ehlers, *Rocket Science for Traders* (John Wiley, 2001).

Before we move on, let's use our TradeStation for two different experiments. The first experiment will deal with the Dynamic Break Out II system and the soybean market. We saw how virtually useless the system was for capturing the trends in the soybean market. What would happen if we faded the trade signals? What we mean by fade is to do just the opposite. So, instead of buying at our long entry point, we will sell and vice versa. If the soybean market moves in cycles, which is countertrend, then we should be able to improve our performance by entering against the prevalent trend. Table 6.5 shows the performance of our countertrend soybean system.

No question that it did better, but overall it is still nothing to write home about. This somewhat proves that soybeans and other grain markets cannot be successfully traded by a longer-term trend-following approach. Since we are on the subject of cycles and seasonality, why don't we program a strategy that incorporates a seasonality filter? We will demonstrate how to use the keyword *date* to determine the current month and day. This system will trade the soybeans and will only take long signals from March 1 to July 1 and will only take

Table 6.5
Soybean Counter Trend Using Dynamic Break Out II

TradeStation Strategy Performance Report—DBSII Fade @S-Daily (6/17/82–4/11/02)			
Performance Summary: All Trades			
Total Net Profit	(1,681.25)	Open position P/L	0.00
Gross Profit	67,031.25	Gross Loss	(68,712.50)
Total # of trades	128	Percent profitable	64.06%
Number winning trades	82	Number losing trades	46
Largest winning trade	6,000.00	Largest losing trade	(11,012.50)
Average winning trade	817.45	Average losing trade	(1,493.75)
Ratio avg win/avg loss	.54725	Avg trade (win & loss)	(13.13)
Max consec. Winners	9	Max consec. losers	3
Avg # bars in winners	11	Avg # bars in losers	32
Max intraday drawdown	(29,043.75)		
Profit Factor	.97553	Max # contracts held	1
Account size required	29,043.75	Return on account	–5.79%

short signals from July 2 to February 28. These dates were derived from cyclical analysis of historical data on soybeans. (Table 6.6 shows the performance of our seasonal soybean system.)

```
Inputs: goLongStart(301),goLongEnd(701),goShortStart(702),goShortEnd(228);
Vars: monthAndDay(0);
{The inputs represent the months and days that we can enter long and short
trades}
{301 is March 01 >> can only go long from this date and up to
701 is July 01 >> this date
702 is July 02 >> can only go short from this date and up to
228 is February 28 >> this date}
{Let's use the date and extract the information that we need from it to
determine the month and the day}
{If we divide the date by 10000, the remainder is the month and day. We can
use the modulus function}
monthAndDay = Mod(Date of tomorrow,10000);
if(monthAndDay >= goLongStart and monthAndDay <= goLongEnd) then
begin
       buy("Seasonal Buy") tomorrow at Open;
end;
```

Table 6.6
Seasonal Soybean System Performance

TradeStation Strategy Performance Report—SeasonalSoybean @S-Daily (4/26/96–4/12/02)			
Performance Summary: All Trades			
Total Net Profit	(4362.50)	Open position P/L	1050.00
Gross Profit	13525.00	Gross Loss	(17887.50)
Total # of trades	11	Percent profitable	36.36%
Number winning trades	4	Number losing trades	7
Largest winning trade	7712.50	Largest losing trade	(5325.00)
Average winning trade	3381.25	Average losing trade	(2555.36)
Ratio avg win/avg loss	1.32320	Avg trade (win & loss)	(396.59)
Max consec. Winners	3	Max consec. losers	4
Avg # bars in winners	145	Avg # bars in losers	120
Max intraday drawdown	(15162.50)		
Profit Factor	.75611	Max # contracts held	1
Account size required	15162.50	Return on account	–28.77%

```
if(monthAndDay >= goShortStart or monthAndDay <= goShortEnd) then
{Notice that we had to use "or" instead of "and"—this is due
to the goShortEnd date is less than the goShortStart date}
begin
       sellShort("Seasonal Sell") tomorrow at Open;
end;
```

THE SUPER COMBO DAY TRADING STRATEGY

So far we have concentrated on longer-term trend-following systems using daily bar analysis. We are now going to work with strategies that deal with intraday bars. Working with intraday bars is the same as working with daily bars. TradeStation doesn't care what time frame you are working with (tick, minute, daily, or weekly); all built-in indicators and functions work the same. The only time programming gets complex is when you are dealing with more than one time frame at a time. Many systems trade on intraday bars, but use daily bars to calculate their buy and sell signals. The Super Combo strategy falls under this category. This system is designed to day trade the stock indices. We should call this the kitchen sink day trader, because we have thrown a lot of different ideas into this one system. This system is rather complex and, therefore, so is the programming code. If you can understand this code, then we doubt there are too many trading ideas that you couldn't program. We will program a system in a modular format to make it easier to understand. In addition, the program code will be heavily laden with comments to further aid in digestion.

Of all of the day trade systems that Futures Truth monitors and ranks, the systems that feature both break out and failed break out technology always tops the list of best performers. Instead of reinventing the wheel, we will borrow this overall concept and use it as a foundation for our own system.

First off, let's discuss the break out component of the Super Combo. If you look at a daily bar on almost any trading instrument, you will notice that the high price is always greater than or equal to the open and close. On non-limit days, it is always higher than the low. At some time during the day, there was more demand than there was supply and, therefore, the market moved up above the open. Conversely, when supply was greater than demand, the market moved down. The objective of an open range break out system is to try and capture some of the market movement between the open and high (or close) or the open and low (or close). The key to successful break out systems is finding the "sweet spot" above or below the open that foretells further movement in that direction. The worst feeling in the world for a day trader is buying at or near the high and selling at or near the low. Your buy point must be above the high of the day when the market has no direction and at the same time must be well below the high when the market has a strong directional move. Your sell point must have the same attributes. You are probably thinking to yourself that this is impossible. You would be right. But we all know that trading is not impossible, difficult yes, but not impossible. This is the beauty of the Super Combo; it knows it must stick its neck out and buy/sell break outs, but it also knows that there is a great likelihood that the initial break out will fail. This is where the failed break out logic kicks into high gear. Once a certain price level is achieved (upside or downside), the system switches gears and begins to look

for failed break out opportunities. How many times have you seen a bar chart where the open is near the high and the close is near the low or just the opposite? These are perfect examples of failed break out opportunities. These concepts are how the Super Combo enters the market.

Once a trade has been initiated, the system uses protective, break-even, and trailing stops to manage the position. The system also utilizes a protective stop reversal. Sometimes a trade is put on and failure occurs within the next couple of hours. This can occur without the system sensing a failed break out. So, under certain conditions, we allow the system to reverse its position at the protective stop level, even when a failed break out isn't signaled. After the system reaches a certain profit level, the protective stop moves to a break-even level. Later in the afternoon, when the likelihood of profit taking and retracements of the overall trend increases, the protective or break-even stops switch to a trailing stop. It may seem to you that the exit techniques are much more complex and thought out than the entry techniques. Many traders don't realize that trade management determines the success of the entry technique and, therefore, they spend all their time trying to figure out the best way to get into the market. Research time should be split between entries and exits. If all goes well with our entry, the position will be closed at the end of trading. There is one more important point concerning this system: no trade can be taken within 30 minutes of the open time. This filter (a technique to help pick out good trades) allows the market to digest any reports that come out before and slightly after the stock indices open. If you take a look at the first 30 minutes of a stock index, especially on report days, you will discover an extreme level of volatility. This volatility will play havoc with any break out-based trading strategy. So, let's just skip it.

Now that we know the overall approach of the Super Combo, let's get a little more specific. The open range break out shouldn't be foreign to you because we described it in the Thermostat strategy. Nor should the buy easier day and sell easier day concept be unfamiliar. The Super Combo system utilizes both ideas to calculate the entry points for the break out component. The first step is to see if we can even take a trade today. We have spent many hours researching the consequences of wide range and small range bars and have drawn the conclusion that small range bars usually give a hint to a subsequent wide range bar. So, if yesterday was a small range bar, then we can attempt a trade today; or else we simply skip today and look for something else to do. We consider a bar to be a small range bar (SRB) by comparing the absolute value of the Open – Close of yesterday's bar to the average of the absolute values of the past ten days' Open – Close values. If the bars' Open to Close range is less than 85 percent of the ten-day average open to close range, then it is considered an SRB. Once we find out that we can trade, we must then determine if today is a buy or sell easier day. A buy easier day occurs when the close is less than or equal to the previous day's close. A sell easier day is simply the con-

verse; today's close is greater than the previous day's close. (Note: We give a slight bias to the buy easier day in case the close of today and yesterday are exactly the same.) Remember, we are making it easier to buy or sell; we are not ruling out a potential buy or sell. Prior to the bear market of 2000, you would be surprised at the number of systems that only bought or made it difficult to sell. If today is a buy easier day, we can buy at the open of today plus 30 percent of the ten-day average true range, and sell at the Open of today minus 60 percent of the ten-day average range (don't use true ranges). On the other hand, if today is a sell easier day, we can sell at the open of today minus 30 percent of the ten-day average range, and buy at the open of today plus 60 percent of the ten-day average range.

Now that we understand the break out component of the Super Combo, let's delve into the failed break out methods. In terms of our system, the following three scenarios define a failed upside break out:

1. The market gaps above yesterday's high a certain amount and then trades below it a certain amount.
2. The market opens below and moves above yesterday's high a certain amount and then trades below it a certain amount.
3. A long position is initiated and failure occurs before 12:00 P.M. central time (CT). Failure is defined as being stopped out. We take advantage of this if failure doesn't happen too quickly.

The opposite is true for a failed break out to the downside. We look to get in the market when any of these three scenarios occur. This type of entry can be defined as countertrend; a position is being entered against the prevailing trend.

When we enter a trade from a break out method, we use a protective stop equal to 25 percent of the average range or three full points, whichever is greater. By using a percentage-based protective stop, we are adapting our protection to the current market condition. If a market is exhibiting high volatility, then a tight fixed stop will get you stopped out with a loss on most trades. On the flipside of the coin, if the market is exhibiting low volatility, then a large fixed stop will risk more than the potential reward and eventually net failure. Adaptive parameters or stops must have boundaries so they won't take on ridiculous values, hence the three-point floor. In today's volatile markets, a protective stop of less than three-points is a disaster waiting to happen. We use the same protective stop on the failed break out entry with one exception. If we enter a position at a stop loss level, we use 15 percent instead of 25 percent of the average range or three full points. The reason behind the different protective stop level for this entry is that there is a chance the market may not have any direction for the entire day. The market proved indecisive by stopping our initial position out. We will nibble on what the market is offering, but we will use a tight stop in case the market performs a double whammie. If the market

is looking kindly on us and provides a profit equal to 50 percent of the 10-day average range, then we pull our protective stop up to a break-even point. In doing so, we achieve a free trade minus slippage and commission. Later in the afternoon, specifically after 2:30 P.M. CT, we trail our protective stop below/above the low/high of the prior 3 five minute bars. Again, through extensive studies, we have discovered that the market will more times than not, fade the overall daily trend in the last 30 to 45 minutes of trading. Losing all of the day's profit in the last fifteen minutes is almost as heartbreaking as buying the high and selling the low. Since we are not trying to scalp the market, we will only test the waters twice on a daily basis. In other words, we will only initiate one long position and/or one short position during the day. Once we have entered the market twice, we stop looking for another entry. That's it. Finito. We have just about given you everything except the kitchen sink. There should be enough ideas in this one system to build a whole slew of different strategies.

The Super Combo sounds rather complicated, doesn't it? If you think it sounds complicated, wait until we try to program it. The best way to attack this monster is one modular block at a time. First, let's pseudocode all of the calculations that deal with daily bars. This may be a good time to introduce TradeStation's capabilities that deal with multiple data streams and time frames. With TradeStation and EasyLanguage, you can analyze up to five different data and/or time frames. In the case of the Super Combo, we will deal with two different time frames: five minute and daily. We will use the daily bar data to feed our calculations and the five-minute data to actually enter our trades. Why five-minute bars instead of fifteen-minute or any other time interval bars? you may ask. If at all possible, it is always best to test your trading ideas on the smallest bar increment that you can. If we had the computer power and time, we would have tested on individual tick bars. The higher the time resolution, the more accurate your testing will be. We will let the online TradeStation help further explain this:

> Calculating Orders on Historical Data Time-based bars include the Open, High, Low, and Close for the specified time period. When you work with historical data, TradeStation doesn't know the chronological order of the transactions that make up the bar. The only transactions for which the chronology is known are the Open, which occurred first, and the Close, which occurred last. With time-based bars based on historical data there is no way to know whether the market opened and then went down, or the market opened and then went up.
>
> However, because the order of ticks can be important, two general rules were established about price movement and the chronological order in which ticks occur.
>
> Assumption 1—The order in which prices on a bar are reached relates to the proximity of the Open to the Low and to the High.

When the Open is closer to the Low than to the High, TradeStation assumes that the Low was reached first. Likewise, if the Open is closer to the High, TradeStation assumes that the High was reached first. For example, say you have a buy order at 100. You protect yourself against losses at 97 points or lower, and want to exit the position at 101 or higher. The Open is 99 points. The High of the day is 103 and the Low is 92, so the Open is closer to the High. In this case, your buy order is filled at 100 and your strategy will generate your profit-taking exit order at 101, thus recording a profit of 1 point.

However, it's possible that in reality the price climbed to 100, at which point your entry order was filled, and then it dipped, hitting 99, before climbing back up to 101. In this case, you would actually have taken a loss of 1 point instead of a profit of 1 point. So, looking at historical data, and based on the market assumption described earlier, TradeStation would record this trade as a winner, when it would in fact have been a loser. The opposite could happen also, where a trade could be recorded as a loser when it was actually a winner.

To more closely simulate market activity in these cases, TradeStation developed what is known as Bouncing Ticks. When enabled, TradeStation automatically sends the data down by a percentage (the TradeStation default is 10%) immediately after a buy or sell order, and then comes back up to the next item on the bar (High, Low, or Close). This action more closely resembles what happens using real-time data.

Using this 10 percent setting on the example above, the buy would have happened at 100, and would have dropped immediately by 10% to 99 (to more closely resemble real-time results), your exit would be recorded at 99, and the bar would more correctly reflect the losing trade that would have happened if you had been using real-time data.

Assumption 2—Symbols trade at every price along the bar.

The second assumption is that a symbol trades at every price along the bar. That may not always be an accurate assumption. If, using the example above again, you had a buy order at 100, and the symbol actually traded at 99 and then jumped to 101, the strategy would record your buy at 100, even though you actually were filled at 101. If your exit was at 110 then, the strategy would record a profit of 10 points, even though your true profit would be only 9 points. This assumption might cause a strategy to appear more or less profitable than it actually was.

TradeStation allows you to avoid both of these assumptions by setting a back-testing resolution. Setting a back-testing resolution allows your strategies to be evaluated according to the actual prices in the order they occurred.

For instructions on enabling the Bouncing Ticks option, see "Setting the Bouncing Ticks Option."

The smaller the time increment that you are dealing with the less there is a chance for error. A fifteen-minute bar is made up of three 5-minute bars. Inside the fifteen-minute bar there may be a price that was never traded; a gap between the high or low and the subsequent open price on a five-minute bar.

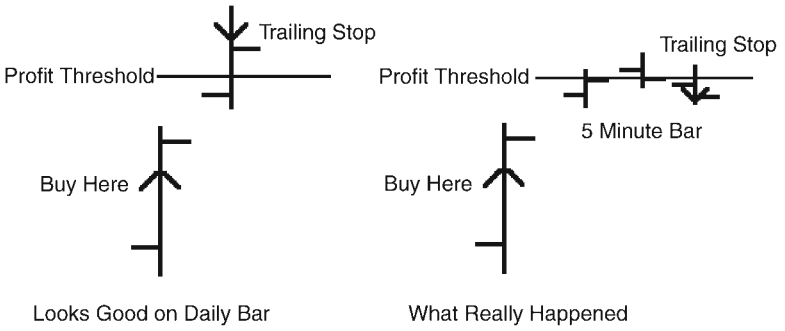

Figure 6.5 Incorrect Daily Bar Assumption

Also, the ambiguity of which occurred first, the high price or the low price, decreases with a smaller time frame. Figure 6.5, which is the same as Figure 4.6, again illustrates how the accuracy of actual market movement increases with smaller time interval bars.

Now that we know why we are using five-minute bars, let's move on. Before we got on the back-testing data resolution tangent, we were about to program the daily bar based calculations.

Super Combo Daily Data Bar Calculation Pseudocode

```
averageRange = Average(Range of Daily Data,10)
averageOCRange = Average(Abs(Open-Close of Daily Data),10)
canTrade = Abs(Open-Close of Daily Data) < 85% of averageOCRange
buyEasierDay = 0
sellEasierDay = 0
if(Close of Daily Data <= Close[1] of Daily Data) then buyEasierDay = 1
if(Close of Daily Data > Close[1] of Daily Data) then sellEasierDay = 1
if(buyEasierDay) then
       buyBOPoint = Open of tomorrow + 30% of averageRange
       sellBOPoint = Open of tomorrow – 60% of averageRange
if(sellEasierDay) then
       sellBOPoint = Open of tomorrow – 30% of averageRange
       buyBOPoint = Open of tomorrow + 60% of averageRange
longBreakPt = High of Daily Data + 25% of averageRange {upside break out
                                                        achieved}
shortBreakPt = Low of Daily Data – 25% of averageRange {dnside break out
                                                        achieved}
longFBOPoint = Low of Daily Data + 25% of averageRange {dnside BO failure
                                                        buy pt}
shortFBOPoint = High of Daily Data – 25% of averageRange {upside BO failure
                                                          sell pt}
```

Now for the 5-minute bar pseudocode. We must initialize some variables on the first bar of the day.

```
if(Date <> Date[1]) then    {a little trick to determine the first bar
                             of the day}
       barCount    = 1
       intraHigh   = 0
       intraLow    = 99999999
       buysToday   = 0
       sellsToday  = 0
       currTrdType = 0
```

EasyLanguage does not have a convenient way to keep track of the intraday high and low, so let's just do it ourselves.

```
if(High > intraHigh) intraHigh = High   {must keep high of current date
                                         manually}
if(Low < intraLow) intraLow = Low       {must keep low of current date
                                         manually}
```

If this bar's date is the same as the previous bar, then we know that we are progressing through today.

```
if(Date = Date[1]) then barCount = barCount + 1
```

If barCount is greater than 6, then we know that we have surpassed the first six bars of the day.

```
if(barCount > 6) then        {skipped first 6 – 5min bars or first 30 minutes}
```

Let's keep track of the number of buy entries and sell entries for today.

```
if(MarketPosition = 1) then buysToday = 1    {if we are long we must have bought
                                              today}
if(MarketPosition = -1) then sellsToday = 1  {same goes for being short}
```

Now let's enter the market on a buy or sell break out.

```
if(buysToday = 0 and Time < 1430) then Buy("LBreakOut") next bar at
   buyBOPoint stop
if(sellsToday = 0 and Time < 1430) then Sell("SbreakOut") next bar at
   sellBOPoint stop
```

Has the market exceeded yesterday's High plus 25 percent of the average range?

```
if(intraHigh > longBreakPoint and Time < 1430) then
```

If it has, look to sell as the market moves back down to yesterday's high – 25 percent of the average range.

```
if(sellsToday = 0) then SellShort("SfailedBO") next bar at shortFBOPoint stop
```

Another failed break out indicator occurs when the market breaks out and gets us into a position and then reverses and stops us out. We can enter a short position on this type of failed breakout if we are stopped out from a long position and at least four 5 minute bars have passed since we entered the long position and it is before 12:00 p.m. central time. We don't allow a reversal if the market immediately stops us out – this is usually a sign of a knee jerk reaction.

```
if(Time < 2300 and sellsToday = 0 and
      longLiqPoint <> EntryPrice and BarsSinceEntry >= 4) then
            SellShort("LongLiqRev") next bar at longLiqPoint stop;
```

Ditto for the failed break out on the short side.

```
if(intraLow < shortBreakPoint and Time < 1430) then
       if(buysToday = 0) then Buy("BfailedBO") next bar at longFBOPoint stop

if(Time < 1200 and buysToday = 0 and
       shortLiqPoint <> EntryPrice and BarsSinceEntry >= 4) then
             Buy("ShortLiqRev") next bar at shortLiqPoint stop;
```

Now this is where it gets a little complicated with the trade management algorithm.

```
if(MarketPosition = 1) then
```

Most of the time, our long liquidation point will be the entry price – 25 percent of the average range or three full points whichever is greater.

```
       longLiqPoint = MinList(EntryPrice – 25% of averageRange,EntryPrice - 3.00
       points) {normal reversal use 25%}
```

However, if we are reversing a short position at a protective stop level then we must use 15 percent. EasyLanguage doesn't have a simple way to determine which filter got you into the current trade. We can figure if the previous position was short and the current position is long and the last bar was the trigger bar and the high of the trigger bar is greater than or equal to our short liquidation point, *and* our short liquidation point is closer to the market than the short failed break out point, then we know we were reversed on to long from a short liquidation point.

```
if(MarketPosition(1) = -1 and BarsSinceEntry(0) = 1 and
High[1]>=shortLiqPoint and shortLiqPoint < shortFBOPoint) then
curTrdType  = -2
if(curTrdType = -2) then
        longLiqPoint = MinList(EntryPrice – 15% of
        averageRange,EntryPrice - 3.00 points)  {long liq reversal use 15%}
```

If we are long and the high of the five-minute bar has exceeded the Entry Price plus 50 percent, then we need to move our stop to break even.

```
if(High > EntryPrice + 50% of averageRange) then longLiqPoint = EntryPrice
```

If time is greater than 2:30 P.M. CT, then we start trailing our long liquidation point from the lowest low of the past 3 five-minute bars.

```
if(time >= 1430) then longLiqPoint = MaxList(longLiqPoint,Lowest(Low,3))
```

Order entry module for either liquidating our long position or initiating a short position at our long liquidation point.

```
if(Time < 1200 and sellsToday = 0 and longLiqPoint <> EntryPrice) and
BarsSinceEntry >= 4) then
        Sell Short ("LongLiqRev") next bar at longLiqPoint stop
else
        Sell ("LongLiq") next bar at longLiqPoint stop
```

Ditto for the short side.

```
If(MarketPosition = -1) then
       shortLiqPoint = MaxList(EntryPrice + 25% of averageRange,EntryPrice +
          3.00 points) {normal reversal use 25%}
       if(MarketPosition(1) = 1 and BarsSinceEntry(0) = 1 and
       (Low[1] <= longLiqPoint and longLiqPoint > longFBOPoint) then
       curTrdType  = +2
       if(curTrdType = +2) then
              shortLiqPoint = MinList(EntryPrice – 15% of
              averageRange,EntryPrice - 3.00 points)
              {long liq reversal use 15%}
       if(Low <= EntryPrice – 50% of averageRange) shortLiqPoint = EntryPrice
       if(Time < 1200 and buysToday = 0 and shortLiqPoint <> EntryPrice) then
              Buy ("ShortLiqRev") next bar at shortLiqPoint stop
       else
              BuyToCover ("ShortLiq") next bar at shortLiqPoint stop
       If(time >= 1430) then shortLiqPoint =
       MinList(shortLiqPoint, Highest(High,3))
```

This next function call gets us out of the market at the closing bell.

```
SetExitOnClose
```

Did you notice how our pseudocode evolved into almost pure EasyLanguage? The more programming that you get under your belt, the less English-like you will be with your pseudocode. You may think you can save time by cutting out this middle step, but you can't. John and I have programmed thousands of systems, and simply outlining the structure and variables ahead of time always saves time in the long run.

Super Combo Code

```
{Super Combo by George Pruitt
This intraday trading system will illustrate the multiple data handling
capabilities of TradeStation. All pertinent buy and sell calculations will be
based on daily bars and actual trades will be executed on 5-min bars. I have
made most of the parameters input variables.}

Inputs:waitPeriodMins(30),initTradesEndTime(1430),liqRevEndTime(1200),
thrustPrcnt1(0.30),thrustPrcnt2(0.60),breakOutPrcnt(0.25),
failedBreakOutPrcnt(0.25),protStopPrcnt1(0.25),protStopPrcnt2(0.15),
protStopAmt(3.00),breakEvenPrcnt(0.50),avgRngLength(10),avgOCLength(10);

Variables:averageRange(0),averageOCRange(0),canTrade(0),buyEasierDay(FALSE),
sellEasierDay(FALSE),buyBOPoint(0),sellBOPoint(0),longBreakPt(0),
shortBreakPt(0),longFBOPoint(0),shortFBOPoint(0),barCount(0),
intraHigh(0),intraLow(999999),buysToday(0),sellsToday(0),
currTrdType(0),longLiqPoint(0),shortLiqPoint(0),yesterdayOCRRange(0),
intraTradeHigh(0),intraTradeLow(999999);

{Just like we did in the pseudocode—let's start out with the daily bar
calculations. If Date <> Date[1]—first bar of day}
if(Date <> Date[1]) then {save time by doing these calculations once per day}
begin
    averageRange = Average(Range,10) of Data2; {Data 2 points to daily bars}
    yesterdayOCRRange = AbsValue(Open of Data2-Close of Data2);
    averageOCRange = Average(AbsValue(Open of Data2-Close of Data2),10);
    canTrade = 0;
    if(yesterdayOCRRange< 0.85*averageOCRange) then canTrade = 1;
    buyEasierDay = FALSE;
    sellEasierDay = FALSE;
    {See how we refer to Data2 - the daily data}
    if(Close of Data2 <= Close[1] of Data2) then buyEasierDay = TRUE;
    if(Close of Data2 > Close[1] of Data2) then sellEasierDay = TRUE;

    if(buyEasierDay) then
    begin
           buyBOPoint = Open of data1 + thrustPrcnt1*averageRange;
           sellBOPoint = Open of data1 - thrustPrcnt2*averageRange;
```

```
    end;
    if(sellEasierDay) then
    begin
           sellBOPoint = Open of data1 - thrustPrcnt1*averageRange;
           buyBOPoint = Open of data1 + thrustPrcnt2*averageRange;
    end;

    longBreakPt = High of Data2 + breakOutPrcnt*averageRange;
    shortBreakPt = Low of Data2 - breakOutPrcnt*averageRange;

    shortFBOPoint = High of Data2 - failedBreakOutPrcnt*averageRange;
    longFBOPoint = Low of Data2 + failedBreakOutPrcnt*averageRange;

{Go ahead and initialize any variables that we may need later on in the day}
    barCount = 0;
    intraHigh = 0;intraLow = 999999; {Didn't know you could do this}
    buysToday = 0;sellsToday = 0;{You can put multiple statements on one
       line}
    currTrdType = 0;
end;   {End of the first bar of data}
{Now let's trade and manage on 5-min bars}
if(High > intraHigh) then intraHigh = High;
if(Low < intraLow ) then intraLow = Low;
barCount = barCount + 1; {count the number of bars of intraday data}
if(barCount > waitPeriodMins/BarInterval and canTrade = 1) then
{have we waited long enough—wait PeriodMin is an input variable and
BarInterval is set by TradeStation. Wait PeriodMins = 30 and BarInterval = 5,
so 30/5 = 6}
begin
    if(MarketPosition = 0) then
    begin
           intraTradeHigh = 0;
           intraTradeLow = 999999;
    end;

    if(MarketPosition = 1) then
    begin
           intraTradeHigh = MaxList(intraTradeHigh,High);
           buysToday = 1;
    end;
    if(MarketPosition =-1) then
    begin
           intraTradeLow = MinList(intraTradeLow,Low);
           sellsToday = 1;
    end;

    if(buysToday = 0 and Time < initTradesEndTime) then
           Buy("LBreakOut") next bar at buyBOPoint stop;
    if(sellsToday = 0 and Time < initTradesEndTime) then
```

```
        SellShort("SBreakout") next bar at sellBOPoint stop;
    if(intraHigh > longBreakPt and sellsToday = 0 and Time <
       initTradesEndTime) then
        SellShort("SfailedBO") next bar at shortFBOPoint stop;
    if(intraLow < shortBreakPt and buysToday = 0 and Time <
       initTradesEndTime) then
        Buy("BfailedBO") next bar at longFBOPoint stop;
{The next module keeps track of positions and places protective stops}
    if(MarketPosition = 1) then
    begin
        longLiqPoint = EntryPrice - protStopPrcnt1*averageRange;
        longLiqPoint = MinList(longLiqPoint,EntryPrice - protStopAmt);
        if(MarketPosition(1) = -1 and BarsSinceEntry = 1 and
            High[1] >= shortLiqPoint and shortLiqPoint < shortFBOPoint)
               then
            currTrdType = -2; {we just got long from a short liq reversal}
        if(currTrdType = -2) then
        begin
             longLiqPoint = EntryPrice - protStopPrcnt2*averageRange;
             longLiqPoint = MinList(longLiqPoint,EntryPrice -
                protStopAmt);
        end;
        if(intraTradeHigh >= EntryPrice + breakEvenPrcnt*averageRange)
           then
             longLiqPoint = EntryPrice; {BreakEven trade}
        if(Time >= initTradesEndTime) then
             longLiqPoint = MaxList(longLiqPoint,Lowest(Low,3)); {Trailing
                stop}
        if(Time < liqRevEndTime and sellsToday = 0 and
             longLiqPoint <> EntryPrice and BarsSinceEntry ≥ 4) then
        begin
             SellShort("LongLiqRev") next bar at longLiqPoint stop;
       end
       else begin
            Sell("LongLiq") next bar at longLiqPoint stop;
       end;
    end;
    if(MarketPosition =-1) then
    begin
         shortLiqPoint = EntryPrice+protStopPrcnt1*averageRange;
         shortLiqPoint = MaxList(shortLiqPoint,EntryPrice + protStopAmt);
         if(MarketPosition(1) = 1 and BarsSinceEntry(0) = 1 and
             Low [1] <= longLiqPoint and longLiqPoint > longFBOPoint) then
             currTrdType = +2; {we just got long from a short liq reversal}
         if(currTrdType = +2) then
         begin
             shortLiqPoint = EntryPrice + protStopPrcnt2*averageRange;
             shortLiqPoint = MaxList(shortLiqPoint,EntryPrice + protStopAmt);
         end;
```

```
        if(intraTradeLow <= EntryPrice - breakEvenPrcnt*averageRange) then
            shortLiqPoint = EntryPrice; {BreakEven trade}
        if(Time >= initTradesEndTime) then
            shortLiqPoint = MinList(shortLiqPoint,Highest(High,3));
               {Trailing stop}
        if(Time < liqRevEndTime and buysToday = 0 and
            shortLiqPoint <> EntryPrice and BarsSinceEntry ≥ 4) then
        begin
            Buy("ShortLiqRev") next bar at shortLiqPoint stop;
        end
        else begin
            BuyToCover("ShortLiq") next bar at shortLiqPoint stop;
        end;
    end;
end;
SetExitOnClose;
```

The Super Combo program demonstrates how to:

- Use the keyword *date* to determine the first intraday bar of the day.
- Create and program a complete trade management scheme.
- Use the keyword *time* to determine optimum trading periods.
- Use and manage multiple data input.
- Exit all positions at the close of the day.

Super Combo trading performance is summarized in Tables 6.7 and 6.8. Since we have a limited amount of historical intraday data to work with, the performance statistics found in Table 6.8 were generated with our Excalibur testing software.

A visual example of how this system enters and exits trades is shown in Figure 6.6.

Super Combo Summary

Wow, this strategy was complicated to program. It wouldn't have been if EasyLanguage had provided a little bit more information on which filter initiated the current trade. The complicated part of the program revolves around determining if we were reversed on a liquidation point and which protective stop to use. We used one liquidation variable for all of our trade management stops; the value of the liquidation variable changed dependent on trade type, time, and open trade profit. If you have simpler exit schemes, you can use the built-in EasyLanguage functions for protective stops and profit objectives. Analyzing different time frames with multiple data streams is much easier than doing the same with only one data stream. You could have done the same thing with

Table 6.7
Super Combo Performance Table 1

TradeStation Strategy Performance Report—SuperCombo @SP-5 min (2/12/2001–4/18/2002) $100 for Slippage and Commission			
Performance Summary: All Trades			
Total Net Profit	27,175.00	Open position P/L	0.00
Gross Profit	126,725.00	Gross Loss	(99,550.00)
Total # of trades	155	Percent profitable	38.06%
Number winning trades	59	Number losing trades	96
Largest winning trade	7,400.00	Largest losing trade	(2,125.00)
Average winning trade	2,147.88	Average losing trade	(1,036.98)
Ratio avg win/avg loss	2.0713	Avg trade (win & loss)	175.32
Max consec. Winners	5	Max consec. losers	10
Avg # bars in winners	48	Avg # bars in losers	20
Max intraday drawdown	(13,725.00)		
Profit Factor	1.2730	Max # contracts held	1
Account size required	13,725.00	Return on account	198.00%

a single five-minute bar chart, but the programming would be considerably more difficult. You could build daily bars from the intraday bars and keep track of all the daily statistics. This could be accomplished by creating array variables and keeping track of when the day begins and ends and the highest and lowest point achieved on an intraday basis. Like we said previously, most traders don't use arrays in their programming, but they are handy when you start doing advanced analysis. For this reason, we have included a strategy that incorporates arrays in Chapter 8.

Back to the subject of using multiple data streams. We have just found that multiple time frame analysis is easier when you can refer to a different data stream for each time frame. Super Combo did relatively well over the past six years. Remember, we haven't done any work with optimizing the many variables in this system. We leave any optimization, tweaking, or application to other markets to you, the reader. The framework of Super Combo is the same framework of many systems that sell for thousands of dollars, so it is up to you to transform this system into your own and make it ten times better.

Table 6.8
Super Combo Performance Table 2

Super Combo Performance from 1986–2/28/2002			
SP500 5 minute bars	**Test 16.17 years**		**4079 trading days**
Total Net P/L	188913$	Avg. Net/Year	11685$
Optimal f	0.19	Geometric Mean	1.003
%Winning Months	59%	Avg. %Ret. MaxDD/Year	39%
		Avg. %Ret. with Time	165%
Max DrawDn ClsTrd	24025$		
Max DrawDn w/OTE	24025$		
Best Trade	10350$	Worst Trade	–2750$
Average Trade	89$	Net Prof:Loss Ratio	1.3
Average Win	981$	Average Loss	–568$
Long Net P/L	104288$	Short Net P/L	84625$
# of Trades	2129	Avg. Trades/Year	132
# of Winning Trade	902	Percent Winners	42.40%
# of Losing Trades	1227	Most Cons. Losses	11
Avg. Days Per Trade	0.5	Longest Flat Time	1077 days
%Time in the Market	24%	Sharpe Ratio	0.25

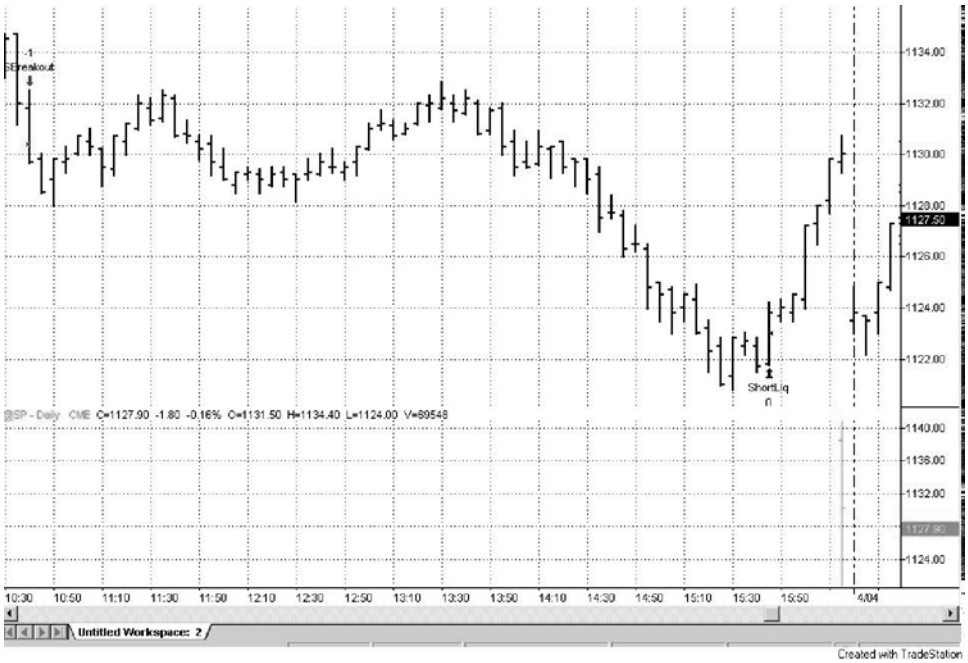

Figure 6.6 Super Combo Trade

THE GHOST TRADER TRADING STRATEGY

The code for the Ghost Trader is designed more as a template than a complete trading strategy. Some trading strategies incorporate the success of the last trade signal in the calculation/determination of the next trade signal. We have tested several methodologies that only initiate new positions after a losing trade. Some traders feel that there exists a high probability of failure on the next trade signal if the last trade was closed out with a profit. This approach is much easier to trade than historically back test. It's easy to keep track of your trades and then skip the hypothetical trades that don't meet your criteria. In our example of waiting for a losing trade before initiating a new trade, you would simply stop trading your account after a winning trade and start paper trading and wait until you have a loser on paper. Once a paper loser occurs, the next trade would be initiated in the real world. The code for Ghost Trader demonstrates how to keep track of simulated trades and only issue the trades that meet a certain criteria.

The core trading strategy of the Ghost Trader is based off of an exponential moving average and a RSI indicator.

```
if(marketPosition = 0 and myProfit < 0 and Xaverage(Close,9) >
   Xaverage(High,19) and
       RSI(Close,9) crosses below 70) then
       begin
              buy next bar at High stop;
       end;
if(marketPosition = 0 and myProfit < 0 and Xaverage(Close,9) <
   Xaverage(Low,19) and
       RSI(Close,9) crosses above 30) then
       begin
              sellShort next bar at Low stop;
       end;
```

Long positions are initiated on the next day at today's high on a stop order. This order is only issued if the nine day exponential moving average of closes is greater than the 19 day exponential moving average of high prices and the nine day RSI of closing prices is crossing from above to below the 70 reading. Short positions are initiated in just the opposite fashion. Long positions are liquidated if today's market action penetrates the lowest low of the past 20 days and short positions are liquidated if today's market action penetrates the highest high of the past 20 days. These entry/exit techniques are interesting but are not the main focus of the Ghost Trader. The main focus of the Ghost Trader is to keep track of a trading system and issue only the trade signals that meet a certain criteria. In our case, actual trade signals should only be issued after a real or simulated losing trade. The following code keeps track of all trades—real and simulated.

Ghost System Code

```
{Ghost system}
{Look to see if a trade would have been executed today and keep track
of our position and our entry price. Test today's high/low price
against the trade signal that was generated by offsetting our calculations
by one day.}
if(myPosition = 0 and Xaverage(Close[1],9) > Xaverage(High[1],19) and
       RSI(Close[1],9) crosses below 70 and High >= High[1]) then
begin
       myEntryPrice = MaxList(Open,High[1]); {Check for a gap open}
       myPosition = 1;
end;
if(myPosition = 1 and Low < Lowest(Low[1],20) )then
begin
       value1 = MinList((Lowest(low[1],20)),Open); {Check for a gap open}
       myProfit = value1 - myEntryPrice            {Calculate our trade
          profit/loss}
       myPosition = 0;
end;
if(myPosition = 0 and Xaverage(Close[1],9) < Xaverage(Low[1],19) and
       RSI(Close[1],9) crosses above 30 and Low <= Low[1]) then
begin
       myEntryPrice = MinList(Open,Low[1]);
       myPosition =-1;
end;
if(myPosition =-1 and High > Highest(High[1],20)) then
begin
       value1 = MaxList((Highest(High[1],20)),Open);{Check again for a gap
          open}
       myProfit = myEntryPrice - value1;    {Calculate our trade profit/loss}
       myPosition = 0;
end;
```

See how we compare today's market action against the signal that was potentially generated on yesterday's bar:

```
if(myPosition = 0 and Xaverage(Close[1],9) > Xaverage(High[1],19) and
RSI(Close[1],9) crosses below 70 and High >= High[1]) then
```

We are checking to see if yesterday's nine day exponential moving average of closes crossed above yesterday's 19 day exponential moving average if highs and yesterday's nine day RSI of closes reading crossed down below 70 and today's high price penetrated yesterday's high price. If all of these criteria were met yesterday and today, then we know we should be in a long position (real or simulated) and the following block of code is executed:

```
begin
       myEntryPrice = MaxList(Open,High[1]); {Check for a gap open}
       myPosition = 1;
end;
```

Notice that we did not instruct TradeStation to issue an order. We are just simply trying to keep track of all trades and we do so by setting our own variables, myEntryPrice and myPosition, accordingly. Remember that we are placing stop orders for the next day at the price of today's high, and since we are keeping track of myEntryPrice, we need to check and see if the Open price of today gapped above our stop price. If the Open did gap above our stop, then we need to manually set myEntryPrice equal to the Open. We also keep track of liquidated trades:

```
if(myPosition = 1 and Low < Lowest(Low[1],20) )then
begin
       value1 = MinList((Lowest(low[1],20)),Open); {Check for a gap open}
       myProfit = value1 - myEntryPrice;    {Calculate our trade profit/loss}
       myPosition = 0;
end;
```

Since TradeStation is not keeping track of all our trades, in addition to myPosition and myEntryPrice, we must keep track of the profit/loss from the last trade by storing the information in myProfit.

Once myProfit is negative (a loss occurred) we can start issuing real trade orders. The following code issues the real signals based on the entry technique and the value of myProfit.

Real System Code

```
{Real System}
{Only enter a new position if the last simulated or real trade was a loser.
If last trade was a loser, myProfit will be less than zero.}

   if(marketPosition = 0 and myProfit < 0 and Xaverage(Close,9) >
      Xaverage(High,19) and
   RSI(Close,9) crosses below 70) then
   begin
          buy next bar at High stop;
   end;
   if(marketPosition = 0 and myProfit < 0 and Xaverage(Close,9) <
      Xaverage(Low,19) and
   RSI(Close,9) crosses above 30) then
   begin
          sellShort next bar at Low stop;
   end;
```

```
if(marketPosition = 1) then sell next bar at Lowest(Low,20) stop;
if(marketPosition =-1) then buytocover next bar at Highest(High,20) stop;
```

The core entry/exit strategy issued the following signals:

```
12/1/2000       Sell    1       .9446   $0.00   $0.00   Short
2/5/2001        SExit   1       .9098   $0.00   $0.00   Cover   4350.0000
3/14/2001       Sell    1       .8674   $0.00   $0.00   Short
5/3/2001        SExit   1       .8563   $0.00   $0.00   Cover   1387.5000
7/3/2001        Sell    1       .8272   $0.00   $0.00   Short
7/20/2001       SExit   1       .8342   $0.00   $0.00   Cover   (875.0000)
8/13/2001       Buy     1       .8433   $0.00   $0.00   Buy
10/11/2001      LExit   1       .8385   $0.00   $0.00   Sell    (600.0000)
10/30/2001      Sell    1       .8323   $0.00   $0.00   Short
11/8/2001       SExit   1       .8443   $0.00   $0.00   Cover   (1500.0000)
11/21/2001      Sell    1       .8250   $0.00   $0.00   Short
3/6/2002        SExit   1       .7669   $0.00   $0.00   Cover   7262.5000
3/11/2002       Buy     1       .7903   $0.00   $0.00   Buy
4/1/2002        LExit   1       .7543   $0.00   $0.00   Sell    (4500.0000)
5/1/2002        Buy     1       .7861   $0.00   $0.00   Buy
```

The core entry/exit technique with the last trade was a loser criteria issued the following trades:

```
12/1/2000       Sell    1       .9446   $0.00   $0.00   Short
2/5/2001        SExit   1       .9098   $0.00   $0.00   Cover   4350.0000
8/13/2001       Buy     1       .8433   $0.00   $0.00   Buy
10/11/2001      LExit   1       .8385   $0.00   $0.00   Sell    (600.0000)
10/30/2001      Sell    1       .8323   $0.00   $0.00   Short
11/8/2001       SExit   1       .8443   $0.00   $0.00   Cover   (1500.0000)
11/21/2001      Sell    1       .8250   $0.00   $0.00   Short
3/6/2002        SExit   1       .7669   $0.00   $0.00   Cover   7262.5000
5/1/2002        Buy     1       .7861   $0.00   $0.00   Buy
```

Over this time period, waiting for a losing trade proved to be much more successful. The Ghost Trader was designed as a template to help in the programming of trading strategies that base the next trade signal off of the results of the prior signal. You could easily modify the code and only issue real trade signals after two consecutive losers.

THE MONEY MANAGER TRADING STRATEGY

An effective trading strategy is only part of a successfully trading plan. If you want your trading to perpetuate, you better have some form of money management built into your overall trading approach. Money management involves examining the concepts of risk and return in reference to investor preference. The objective is to choose a desired rate of return and then minimize the risk associated with that rate of return. Money management concepts should be used to make the most efficient use of trading capital. We can't emphasize enough the importance of using money management in a trading plan. The Money Manager strategy is a simple system that incorporates and demonstrates some simple money management concepts. The concepts we are presenting go beyond simple profit objectives or protective stops. These ideas fall within the realm of the underlying trading strategy. We go beyond this and move into the areas of capital allocation. The concepts that are presented in this strategy are based on capital preservation and market normalization. We all know what capital preservation is, but some may not understand the concept of market normalization. The ability to diversify equal amounts of capital across a portfolio of different markets is the backbone of any money management scheme. If we want to risk 5% of our equity on soybeans and 5% on Treasury bonds, we need the ability to treat the two markets on apples to apples basis. Most of the time one contract of Treasury bonds exhibits more risk than one contract of soybeans. Since we want to maintain a constant amount of capital to risk on the two markets, we will need to trade less Treasury bonds and more soybeans. Let's say the implied market risk for Treasury bonds is $1000 and $500 for soybeans. If we were risking $2000 on each market, we would then trade 2 contracts of bonds and 4 contracts of soybeans. We are maintaining the same amount of risk by varying the number of contracts for the two markets.

Measuring market risk is the first step in the market normalization process. Money managers use several different measures to monitor market risk: average true range, mean change in closing prices, standard deviation in closing prices, and numerous others. The Money Manager strategy uses the standard deviation in closing prices to calculate market risk.

```
Inputs: initCapital(100000),rskAmount(.02);
Vars: marketRisk(0),numContracts(0);

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

The StdDev function returns the standard deviation in terms of points, so we multiply by BigPointValue (dollar value of a big point move) to get market risk in terms of dollars. Once we know the market risk, we then can calculate the number of contracts.

```
numContracts = initCapital * rskAmount / marketRisk;
```

For demonstration purposes, let's assume we are trading the Japanese Yen and the market risk is equal to $750. The number of contracts would be calculated by using the formula from above:

```
numContracts = 100000 * .02 /750
numContracts = 2000/750
numContracts = 2.66667
```

Since we can only trade with whole contracts, we round down to the nearest whole number. Since market risk is the denominator in our formula, whenever market risk increases the number of contracts decrease, hence, the risk aversion component of our money management scheme. In similar fashion to the Ghost Trader, the Money Manager was designed as more of a template than an actual trading strategy. We wanted to provide the tools necessary to build a money management platform. The source code for Money Manager follows.

The Money Manager Code

```
{The Money Manager}
{Demonstrates the programming and use of a money management scheme.}
{The user inputs initial capital and the amount he wants to risk on each
trade.}
Inputs: initCapital(100000),rskAmt(.02);
Vars: marketRisk(0),numContracts(0);
marketRisk = StdDev(Close,30) * BigPointValue;
numContracts = (initialCapital * rskAmt) / marketRisk;
value1 = Round(numContracts,0);
if(value1 > numContracts) then
       numContracts = value1 - 1
else
       numContracts = value1;
numContracts = MaxList(numContracts,1); {make sure at least 1 contract is
                                         traded}

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

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

Overall the logic should be easy to follow. However, there is some code that may not be totally intuitive. The formula that we used to determine the number of contracts doesn't always produce a whole number. Since we are trading

stocks or futures, we can't have a fractional number of contracts (shares). The Round function was used to eliminate the fractional part. The Round function requires two parameters: the value to be rounded and the level of precision. The level of precision parameter relates to how many decimal places are to be rounded to. If we had used one instead of zero, then the number of contracts would have been rounded to the nearest tenth place. We used zero to round to the nearest whole number. If we had passed two as the parameter, then the function would round to the nearest 100th place. Since we developed a risk averse money management model, the number of contracts should never be rounded up (risk exposure is directly proportional to the number of contracts that we are trading). EasyLanguage does not provide a truncation function (a function that eliminates the fractional part of a number), so we used the following code to always round down:

```
value1 = Round(numContracts,0);
if(value1 > numContracts) then
        numContracts = value1 - 1
else
        numContracts = value1;
```

Value1 is assigned numContracts rounded to the nearest integer. If the round function rounds up, then value1 will be greater than the numContracts variable. If value1 is greater than numContracts, we simply subtract one from value1 and reassign the difference to numContracts. If value1 is not greater than numContracts, then the round function rounded down and we can simply reassign numContracts this value.

We introduced the keyword shares in the order placement logic.

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

The keyword *shares* must be used when trading a variable number of contracts or shares. The number of shares (in the form of a variable name or a literal) must follow the Buy/SellShort keyword and precede the keyword shares.

```
Buy("myBuy") 50 shares tomorrow at Open;
```

Or

```
Buy("myBuy") myNumShares shares tomorrow at Open;
```

You can use Money Manager as a platform to further research into other money management schemes. You can get some great money management ideas out of Nauzer J. Balsara's, "Money Management Strategies for Futures Traders", published by John Wiley and Sons in 1992.

CONCLUSIONS

Serious programming was introduced as well as some robust trading system principles: Bollinger Bands, Keltner Channels, Donchian Break Outs, Open Range Break Outs, Day Trading, and the usage of multiple time frames, as well as Ghost Trader and Money Manager Trading Strategies. Of course, these aren't the only trading principles available. There are many other existing principles and some that have yet to be uncovered. The search for market principles is what keeps a system trader, researcher, and programmer excited about what he is doing. Over the next few chapters, we will try and introduce a few more morsels of research. The next chapter will focus on the art of debugging—the ugly side of programming.

7

Debugging and OutPut

Throughout this book, we have compared EasyLanguage to some of today's powerful programming languages. The power of the language is somewhat diminished by the integrated development environment (IDE) in which it is encapsulated. Today's contemporary programming environments usually consist of three elements: (1) editor, (2) compiler, and (3) in-line debugger. TradeStation has two out of the three. Guess which one is missing. If you guessed debugger then you are absolutely correct. You have already seen the editor and compiler in action. If you are coming from a nonprogramming background, this may not seem like a big deal. First off, let us explain what a debugger is. A debugger is a program that allows a user to step through each individual line of code and evaluate the program statements as they are executed. Why on earth would you want to do this? you may ask. Unfortunately, most programs don't work properly the instant you finish typing them. Even though your program may compile or verify, this doesn't mean it will work in the way you intended. Usually the most difficult part of the process of programming starts after you have carefully typed your ideas into the PowerEditor and verified your program. The Super Combo system that we programmed in the last chapter required about an hour to pseudocode, an hour to type the program in, and a couple of hours to debug. Even with careful planning and typing, we didn't initially get the Super Combo to work in the manner that we intended. The code that you saw in the last chapter was the result of trial and error.

This chapter discusses topics associated with the process of fixing your program code to accurately reflect your trading ideas. In addition, we will also discuss the tools that can be used to create customized reports.

LOGICAL VERSUS SYNTAX ERRORS

As you verify your analysis technique, TradeStation pauses to look for syntax errors in your code. If there are any syntax errors, TradeStation opens up a dialog box explaining the error and places the cursor on the offending piece of code. You can fix the error and then reverify and then move on to the next error, if one exists. If you like to see more detailed information about your syntax errors, then prior to verifying your code, open up the EasyLanguage output bar and select the Verify tab. (You do this by going under the View menu and selecting EasyLanguage Output Bar.) This window will tell you a description of the error, the analysis technique that caused it, the offending line number, and the type of error. We always verify with this window open. These syntax errors are tedious and time-consuming, but are relatively easy to correct. We give a large list of the more popular syntax errors in Appendix A.

There are basically two types of errors in programming: syntax and logical. The logical errors are sometimes quite difficult to fix and are usually the ones that instigate the violent action of throwing one's monitor through the closest open or shut window. A logical error is an imperfection in your translation of your ideas into programming code. This error may simply be a typo or a complete misunderstanding of a concept. A typo is a simple fix, whereas a flawed conception of an idea may take hours to work through. A debugger is the most common cure for the logical error. Alas, we don't have a built-in debugger. Never fear, neither did the programming pioneers. We can get around this obstacle with the use of cleverly placed print statements and the Print Log. The Print Log can be accessed through the EasyLanguage Output bar by selecting the Print Log tab.

DEBUGGING WITH THE PRINT STATEMENT AND PRINT LOG

The Print Log and Print Statement aren't perfect, but they should be an acceptable substitute for a debugger. The following snippet of code has a logical error; see if you can visually find it.

```
{Protective stop logic}
longLiqPoint = EntryPrice + 1.5*Range;
shortLiqPoint = EntryPrice + 1.5*Range;
if (MarketPosition = 1) then sell next bar at longLiqPoint stop;
if (MarketPosition) = -1) then buy to cover next bar at shortLiqPoint stop;
```

If you found the error, then let's keep it to ourselves and continue on as if we didn't. If we had actually typed in this code, it would have verified with no problems. The problems wouldn't arise until we applied the logic to a chart

and started to analyze the trades. After looking at the trades, you will notice that there is something wrong with our long liquidation logic. Let's push our sleeves up and delve in and debug it by adding a few lines of code.

```
Vars: longLiqPoint(0),shortLiqPoint(0);
{System uses William's Percent R indicator to determine OB/OS situations}
if(percentR(14) crosses above 20) then buy next bar at
       Highest(High,3) stop;
if(percentR(14) crosses below 80) then sellShort next bar at
       Lowest(Low,3) stop;
longLiqPoint = EntryPrice + 1.5 * Range;
shortLiqPoint = EntryPrice + 1.5 * Range;
{Debugging Starts Here!}
if(MarketPosition = 1) then print(Date,EntryPrice,longLiqPoint);
{Debugging Ends Here!}
if(MarketPosition = 1) then sell next bar at longLiqPoint stop;
if(MarketPosition = -1) then buyToCover next bar at shortLiqPoint stop;
```

The Print Statement can print any variable out to the Print Log. Here we are printing the current date, the entry price for our long position, and the liquidation price. Once you verify the program with the new debugging code and apply it to a Microsoft chart, the information will be output to the Print Log and look like the information in Table 7.1.

Table 7.1
Print Log Output

1000808	71.13	78.17
1000829	72.13	74.93
1001019	58.44	64.72
1001212	58.75	63.63
1001226	47.13	51.15
1010104	48.88	54.31
1010226	58.06	61.9
1010323	56.13	60.06
1010522	70.14	71.96
1010621	69.59	72.04
1010827	62.28	64.96
1010925	52.61	56.87
1011130	65.08	66.81
1020103	68.85	72.09
1020201	64.5	67.54
1020409	57.33	61.28

Output from debug code.

If you examine the information in the Print Log, you'll notice something wrong with our long liquidation stop; it's above the entry price. We intended this stop to be a protective stop and not a profit objective. This problem can be fixed simply by changing the "+" to a "–" in the line of code that sets the longLiqPoint.

We wish all logical errors were this simple. The problem is finding the simple errors. If you have no idea where a logical error may be located, then you basically have to print out the values of all of your variables on a bar-by-bar basis. We have worked on programs where the number of Print Statements outnumbered the actual program statements two to one. The larger and more complex a program is, the more difficult and tedious the debugging will be. This is yet another reason to program in a modular format. If you have a complicated program that consists of several different modules, then program and debug each module one at a time. Program and debug module A and then move on to module B and so on. In doing so, if a logical error does creep into your code, then you will know its relative location.

TABLE CREATOR

The following code demonstrates how we used Print Statements to locate our logical errors and at the same time create a report. Before we go into the actual code, let us first explain what the program is attempting to do. We thought it would be interesting to do research on the relationship of the opening price and closing price to different zone levels. We got the idea for this type of research from *Detecting High Profit Trades in the Futures Markets* by J. T. Jackson, published in 1994 by Windsor Books. We cut today's bar into four different zones: Zone1 is anything greater than today's high; Zone2 is anything less than or equal to today's high and greater than or equal to today's midpoint; Zone3 is anything less than today's midpoint and greater than or equal to today's low; and Zone4 is anything less than today's low. Our program will count the number of times the market opens in the four zones and keep track of the relationship between the opening and closing zones. In other words, we are trying to find the probability of the market opening in Zone1 and closing in Zone1, opening in Zone1 and closing in Zone2, and so on. In the end, the program will generate a report that looks something like the following:

```
        Close   Zone1   Zone2   Zone3   Zone4
Open
Zone1           25%     18%     17%     40%
Zone2           –       –       –       –
Zone3           –       –       –       –
Zone4           –       –       –       –
```

```
{Table Creator by George Pruitt
We knew this program was going to be complicated, so we began with setting
up a debugging framework to start. This program isn't a simple strategy, but
instead a platform for research and analysis. We wanted to set up a
probability study based on the relationship between the opening and closing
prices and different zone levels.}

Vars: traceOn(FALSE),debugOn(FALSE),createReport(FALSE);
Vars: zone1(0),zone2(0),zone3(0);
Vars: inZone1(0),inZone2(0),inZone3(0),inZone4(0);
Vars: inZone1.outZone1(0),inZone1.outZone2(0),inZone1.outZone3(0),
              inZone1.outZone4(0);
Vars: inZone2.outZone1(0),inZone2.outZone2(0),inZone2.outZone3(0),
              inZone2.outZone4(0);
Vars: inZone3.outZone1(0),inZone3.outZone2(0),inZone3.outZone3(0),
              inZone3.outZone4(0);
Vars: inZone4.outZone1(0),inZone4.outZone2(0),inZone4.outZone3(0),
              inZone4.outZone4(0);
Vars: myBarCounter(0);
{Module 1 starts here!}

debugOn = TRUE; {This turns the flag on to print out various debug info}
traceOn = FALSE; {This turns the flag on to print out almost all statements}
createReport = TRUE; {Print the report to the print log}
{Start the analysis with the calculation of the different zones}

zone1 = High[1];
zone2 = High[1] + Low[1]/2.0;
zone3 = Low[1];
{Here we will print out the elements of our zone calculations to assist with
our debugging}
if(debugOn) then print(Date:8:0,High[1],Low[1],High[1] + Low[1]/2.0);
{If we turn trace on, then the program will dump a ton of information}
if(traceOn) then
begin
       if(BarNumber = 1) then print("**Start Trace",Date:6:0," **");
       print(" Zones for ",Date:8:0);
       print(" Zone1 = ",zone1);
       print(" Zone2 = ",zone2);
       print(" Zone3 = ",zone3);
end;
{!!!! Module 1 ends here}
myBarCounter = myBarCounter + 1;

if(Open > zone1) then {This begins opening in Zone 1 Module}
begin
inZone1 = inZone1 + 1;
if(Close > zone1) then inZone1.outZone1 = inZone1.outZone1 + 1;
```

```
if(Close <= zone1 and Close > zone2) then
       inZone1.outZone2 = inZone1.outZone2 + 1;
if(Close <= zone2 and Close > zone3) then
       inZone1.outZone3 = inZone1.outZone3 + 1;
if(Close <= zone3) then inZone1.outZone4 = inZone1.outZone4 + 1;
if(debugOn) then {Here we print the stats for an open in zone1}
begin
       print("Open in Zone1 ",Date :6:0,zone1,Open,Close,inZone1,
       inZone1.outZone1,inZone1.outZone2,inZone1.outZone3,
       inZone1.outZone4);
end;
end; {This concludes opening in Zone1 Module}
```

At this point in the program, we would take a break from programming and verify and apply the logic to a Microsoft chart. We would examine the output in the Print Log to make sure that we had programmed everything accurately. Table 7.2 shows the output of our initial programming.

Remember the statement:

```
if(debugOn) then print(Date :8:0,High[1],Low[1],High[1] + Low[1]/2.0);
```

We print out the date, high of yesterday, low of yesterday, and the midpoint of yesterday (Zone1, Zone3, Zone2, respectively). Do you see something wrong in the print out? It looks like Zone 2 is greater than Zone 1. Our debugging already caught its first bug. Remember, Zone 2 is the area between and including today's midpoint and today's high price. According to our print out, Zone 2 is above Zone 1. Let's take a look at our Zone 2 calculation:

```
zone2 = High + Low/2.0;
```

What's wrong with this? Argh! We don't want this. We want the midpoint or the average of today's extreme prices:

```
zone2 = (High + Low)/2.0;
```

Remember the order of operations? Division is done first and the low is first divided by 2.0 and then added to the high. We must use parentheses to get the correct order. It was a good thing we caught this right off the bat, or it may have caused several headaches down the road. Now back to the program. Let's program for the occurrence of the open between Zone 1 and Zone 2.

```
if(Open <= zone1 and Open >= zone2) then
begin
       inZone2 = inZone2 + 1;
       if(Close > zone1) then inZone2.outZone1 = inZone2.outZone1 + 1;
       if(Close <= zone1 and Close > zone2) then
              inZone2.outZone2 = inZone2.outZone2 + 1;
```

Table 7.2
Debug Output

1020306	63.88	62.34	95.05					
1020307	63.7	62.19	94.79					
1020308	63.89	61.86	94.82					
1020311	64.7	63.17	96.28					
1020312	65	63	96.5					
1020313	62.8	61.67	93.63					
1020314	63.02	61.95	94					
1020315	62.24	61.05	92.76					
1020318	62.51	60.97	92.99					
Open in Zone	1020318	62.51	62.74	62.14	40	23	0	38	2
1020319	62.99	61.2	93.59					
1020320	63	61.5	93.75					
1020321	62.02	60.1	92.07					
1020322	61.6	59.83	91.51					
1020325	61.14	60.22	91.25					
1020326	60.78	59.15	90.35					
1020327	60.92	58.31	90.07					
1020328	59.88	58.59	89.18					
Open in Zone	1020328	59.88	59.95	60.31	41	24	0	39	2
1020401	60.65	59.66	90.48					
1020402	60.4	59.2	90					
1020403	59.1	57.11	87.65					
1020404	57.61	55.5	85.36					
1020405	56.97	55.43	84.68					
1020408	57.3	55.84	85.22					
1020409	57.31	54.26	84.44					
Open in Zone	1020409	57.31	57.33	54.87	42	24	0	40	2
1020410	57.43	54.8	84.83					
1020411	57.11	55.06	84.64					
1020412	56.45	54.5	83.7					
1020415	56.26	54.86	83.69					
1020416	56.77	55.4	84.47					
1020417	58.1	56.36	86.28					
1020418	58.28	56.42	86.49					

Print out of Print Log from TableCreator program.

```
        if(Close <= zone2 and Close > zone3) then
                inZone2.outZone3 = inZone2.outZone3 + 1;
if(Close <= zone3) then inZone2.outZone4 = inZone2.outZone4 + 1;
if(debugOn) then {Here we print the stats for an open in zone2}
begin
                print("Open in Zone1", zone1Open, Close,inZone2,
                inZone2.outZone1,
                inZone2.outZone2, inZone2.outZone3,inZone2.outZone4);
        end;
end;
{!!!!! Module 2 ends here and Module 3 starts here}
```

For brevity's sake, we will skip the other two zone calculations and move on to creating the probability analysis table.

```
If(LastBarOnChart and createReport) then
begin
        Print("Close "," Zone1 Zone2 Zone3 Zone4");
        Print("-------------------------");
        Print("Open |");
        {How many times did we open in zone1 and close in zone1}
        value1 = inZone1.outZone1/inZone1*100; {value1 is a temporary
           variable}
        value2 = inZone1.outZone2/inZone1*100; {value2 is a temporary
           variable}
        value3 = inZone1.outZone3/inZone1*100; {and so on}
        value4 = inZone1.outZone4/inZone1*100;
        Print("Zone1 | ",inZone1,value1:3:0,"% ", value2:3:0,"% ",
           value3:3:0,"% ", value4 :3:0,"%");
        value1 = inZone2.outZone1/inZone2*100; {value1 is a temporary
           variable}
        value2 = inZone2.outZone2/inZone2*100; {value2 is a temporary
           variable}
        value3 = inZone2.outZone3/inZone2*100; {and so on}
        value4 = inZone2.outZone4/inZone2*100;
        Print("Zone2 | ",inZone3,value1:3:0,"% ", value2:3:0,"% ",
           value3:3:0,"% ", value4 :3:0,"%");
```

Again for brevity, we will skip the remaining portion of the report creation code. Don't worry, you will find the program in its entirety on the enclosed CD-ROM. We are sure you will find the research quite interesting. We do include the probability table of the S&P 500 at the end of this chapter; the exact one generated by this code.

Remember when we said that the Super Combo strategy was the end result of trial and error and a couple of hours of debugging? Take a look at a portion of the code that dealt with setting the longLiqPoint with our debug statements.

```
{The next module keeps track of positions and places protective stops}
if(MarketPosition = 1) then
begin
       longLiqPoint = EntryPrice - protStopPrcnt1*averageRange;
       print("1 Long",longLiqPoint,entryPrice);
       longLiqPoint = MinList(longLiqPoint,EntryPrice - protStopAmt);
       print("2",longLiqPoint);
       if(MarketPosition(1) = -1 and BarsSinceEntry = 1 and
              High[1] >= shortLiqPoint and shortLiqPoint < shortFBOPoint)
                 then currTrdType = -2; {we just got long from a short liq
                 reversal}
          if(currTrdType = -2) then
       begin
              longLiqPoint = EntryPrice - protStopPrcnt2*averageRange;
              longLiqPoint = MinList(longLiqPoint,EntryPrice - protStopAmt);
              print("3",longLiqPoint,currTrdType,barssinceentry(0));
       end;
       if(High >= EntryPrice + breakEvenPrcnt*averageRange) then
              longLiqPoint = EntryPrice; {Break-even trade}
       print("4",longLiqPoint);
       if(Time >= initTradesEndTime) then {Trailing stop}
              longLiqPoint = MaxList(longLiqPoint,Lowest(Low,3));
          print("5",longLiqPoint);
       if(Time < liqRevEndTime and sellsToday = 0 and
              longLiqPoint <> EntryPrice) then
       begin
              SellShort("LongLiqRev") next bar at longLiqPoint stop;
       end
       else begin
              Sell("LongLiq") next bar at longLiqPoint stop;
       end;
end;
```

See how we printed the value of the protective stop each and every time it was changed. We printed a different number for each condition that could change the protective stop and the actual value of the protective stop. Even professional programmers have to debug. Heck, most of them use debugging as a programming tool and not a fix. The truth of the matter is that most programmers (EasyLanguage or whatever language) do very little planning. They start coding right off the bat. If they don't know how a function works or the correct keyword, they will output the results to a debugger to find out. In most cases, they spend way too much time in front of the computer. We aren't looking down our noses at these programmers, because we do the same thing. However, if you can learn the correct way and be disciplined enough, you can considerably decrease debugging time and prevent your monitor from flying out a window. Well, enough with the lecture.

Table 7.3
Zone Report

Close	Zone1	Zone2	Zone3	Zone4
Open				
Zone1	61%	23%	8%	8%
Zone2	33%	34%	19%	13%
Zone3	14%	24%	31%	31%
Zone4	9%	12%	21%	58%

Output of the Print Log that was created with TableCreator.

The Print Statement can be used to print out to a text file. The probability table that we created could be printed to a file instead of the Print Log. To do this, you would simply insert the name of the file into the existing Print Statements:

```
Before:        Print(" Close "," Zone1 Zone2 Zone3 Zone4");
After:         Print("c:\MyFile"," Close "," Zone1 Zone2 Zone3 Zone4");
```

This Print Statement will print to MyFile on the C: drive. Each time you apply your code with this Print Statement to a chart, MyFile will be deleted and recreated and printed to. After the analysis technique is applied, the file can then be opened with a word processor, emailed to others, copied to a disk, or anything else you can do with a file. This is a powerful tool for creating reports and analysis. Table 7.3 is a sample of the output that we created with our CreateReport program.

CONCLUSIONS

Debugging is an essential part of programming. Without it, how would we fix our logic and our programs? The Print Statement can be used to print out the values of our variables to the Print Log or a file. You can also format the output of the print statement like in the following:

```
myValue1 = 132.4444455;
Format                          Output
Print(myValue1 :4:2);           132.45
Print(myValue1 :4:0);           132
Print(myValue1 :4.1);           133.5

Printing to a file is as easy as printing to the print log:
Print("C:\myDataFile",myValue1);
```

This statement prints myValue1 to myDataFile on the C: drive. Each time a new Print Statement is encountered a line feed is generated. So for every print statement, you will have a line of output. The file is destroyed and recreated every time a new bar is added to the chart, the program is verified, or the program is applied to a new chart.

You will soon discover how helpful the Print Statement really is. Without it, we don't think there is any way you could become a productive EasyLanguage programmer. So far, our discussions have mostly revolved around programming strategies. The next chapter shifts gears and we concentrate on the research and analysis capabilities of EasyLanguage and TradeStation.

8

TradeStation as a Research Tool

Before you can create a trading strategy, you must develop and research a trading idea. A strategy starts out as an initial hunch, "What if I buy when the market makes a new five-day high and the ADX is greater than 40?" The life of this trading idea is either ended quickly or extended through research. If we program an idea and test it and it fails miserably, then we usually file it in the circular file. However, if the idea shows some potential, then we go on from there. The next step is adding some other ideas to the core; "What if I force a 19 bar Stochastic reading to be in the oversold region in addition to a new five-day high and the ADX reading is greater than 40?" This is usually how the developmental cycle of a trading strategy works. In this chapter, we will show how to use TradeStation, EasyLanguage, and optimization as tools for developing, researching, and testing trading ideas. These ideas may or may not evolve into complete trading strategies. The ideas that we will expound upon involve the use of external data, pattern recognition, and intermarket analysis. We hope the ideas and programming that we present in this chapter will provide a sound foundation for you to build your research tools.

COMMITMENT OF TRADERS REPORT

Many of the trading strategies and ideas that we have tested over the past fifteen years solely relied on price data. However, there are trading ideas and strategies that incorporate data outside of market prices. The next programming project will demonstrate how to include external nonprice data in your

testing. The Commodity Futures Trading Commission's (CFTC) *Commitment of Traders* report is an example of useful nonprice specific data. This report summarizes the positions held by reporting and nonreporting traders. If a trader's position exceeds a certain level, it must be reported to the CFTC. This reporting procedure prevents any one individual or group from controlling or cornering the market. The COT report subdivides the open interest (all positions held) into commercial (hedgers) interest and speculative interest. The commercial and speculative positions are then divided into long and short positions. This report was originally released on the 11th day of the month. Since 1992 it has been released on a biweekly basis. We thought it would be interesting research to find out if the commercial interest had any insight in the direction of the market. If the commercial positions were net bullish, we would buy. If the commercial positions were net bearish, we would sell. Initially, when we thought about this, we thought it would be an easy test. The idea was easy enough, but the processing of the data and feeding it into TradeStation wasn't. The *Commitment of Traders* report can be accessed through the CFTC website <www.cftc.gov>. The daily report looks like the one in Table 8.1.

As you can see, the report is easy enough for the human eye to read. However, the computer can't interpret all of this information without first writing a sophisticated parsing program. Of all of the information that is presented in the report, we are only interested in the long and short positions held by the commercials. So, we downloaded each report, extracted the necessary information, and compiled it into a streamlined report (see Table 8.2).

After we compiled the information, the problem then was to get it into TradeStation. Data importation is not a big deal in any version of TradeStation prior to 6.0. Those versions allowed the importation of third-party data. You could create an ASCII file with date, open, high, low, and close fields delimited by a space or a comma. Once you transformed your data into this rather flexible format, you could simply insert the data into a chart and refer to it in your EasyLanguage program in a similar fashion as we did with the Super Combo system. One of the benefits of TradeStation 6.0 is also one of its biggest drawbacks: a standardized single data provider. You are limited to the type and amount of data that is provided by this one source. Don't get us wrong, we are very impressed with the amount of data that is provided. You have access to stocks, futures, options, indices, and mutual funds. The information that we are looking for, in this particular project, is somewhat unusual and very customized. Unfortunately for TradeStation 6.0 users, the synthesis of data and software in one complete package does not allow highly customizable solutions. Earlier versions of TradeStation were much more open and scalable. The current, not so open, platform makes this type of research very difficult. However, as we did with our debugger, we can also overcome this obstacle. Our solution is not a pretty one and requires a lot of elbow grease.

Table 8.1
Committment of Traders Report

S&P 500 STOCK INDEX - CHICAGO MERCANTILE EXCHANGE
COMMITMENTS OF TRADERS IN ALL FUTURES COMBINED AND INDICATED FUTURES, April 9, 2002

FUTURES	TOTAL OPEN INTEREST	REPORTABLE POSITIONS: NON-COMMERCIAL LONG OR SHORT ONLY: LONG	NON-COMMERCIAL LONG OR SHORT ONLY: SHORT	NON-COMMERCIAL LONG AND SHORT (SPREADING): LONG	NON-COMMERCIAL LONG AND SHORT (SPREADING): SHORT	COMMERCIAL: LONG	COMMERCIAL: SHORT	TOTAL: LONG	TOTAL: SHORT	NONREPORTABLE POSITIONS: LONG	NONREPORTABLE POSITIONS: SHORT
	(S&P 500 INDEX X $250.00)										
ALL	505,813	30,176	42,761	4,299	4,299	320,101	411,075	354,576	458,135	151,237	47,678
OLD	505,813	30,176	42,761	4,299	4,299	320,101	411,075	354,576	458,135	151,237	47,678
OTHER	0	0	0	0	0	0	0	0	0	0	0
	CHANGES IN COMMITMENTS FROM April 2, 2002										
ALL	9,790	905	223	290	290	6,807	4,738	8,002	5,251	1,788	4,539
	PERCENT OF OPEN INTEREST REPRESENTED BY EACH CATEGORY OF TRADERS										
ALL	100.0%	6.0	8.5	0.8	0.8	63.3	81.3	70.1	90.6	29.9	9.4
OLD	100.0%	6.0	8.5	0.8	0.8	63.3	81.3	70.1	90.6	29.9	9.4
OTHER	100.0%	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

	NUMBER OF TRADERS	NUMBER OF TRADERS IN EACH CATEGORY							
ALL	138	18	19	10	10	80	59	106	80
OLD	138	18	19	10	10	80	59	106	80
OTHER	0	0	0	0	0	0	0	0	0

CONCENTRATION RATIOS

PERCENT OF OPEN INTEREST HELD BY THE INDICATED NUMBER OF LARGEST TRADERS

	BY GROSS POSITION				BY NET POSITION			
	4 OR LESS TRADERS		8 OR LESS TRADERS		4 OR LESS TRADERS		8 OR LESS TRADERS	
	LONG	SHORT	LONG	SHORT	LONG	SHORT	LONG	SHORT
ALL	21.6	31.6	30	45.4	21.3	30.8	29.5	44.5
OLD	21.6	31.6	30.2	45.4	21.3	30.8	29.5	44.5
OTHER	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 8.2
Simplified Version of the *Committment of Traders* Report

Date	Bulls	Bears
19860115	29116	29119
19860131	31665	27329
19860214	37974	32357
19860228	42672	41513
19860314	44976	60318
19860331	31899	48075
19860415	29808	38246
19860430	28780	38111
19860515	52436	43043
19860530	53255	49183
19860613	55666	46562
19860630	45747	46670
19860715	57792	52414
19860731	63517	62979
19860815	54968	50903
19860829	58622	52918
19860915	75942	84017
19860930	90072	73655
19861015	96932	77658
19861031	98417	82308
19861114	100415	72907
19861128	90516	76338
19861215	74731	72098
19861231	58122	57665
19870115	67508	50323

The perfect solution would be the ability to import data from a text file as was available on previous versions of TradeStation. Since we don't have this luxury, we will type the data that we need into array data structures. Remember, arrays are lists of similar data that can be accessed with an index variable. In this program, we will use three parallel arrays. We hope this doesn't sound too computer geeky. This research project doesn't require any sophisticated array manipulations. Maybe we should use the word *table* instead of *parallel arrays*. Our table will look something like:

Date	Commercial Bulls	Commercial Bears
20011016	378866	415289
20011023	377177	413658
20011030	377468	413729
20011106	376807	416063
20011113	381539	421284
20011120	369784	415822
20011127	371336	421405
20011204	360315	420919
20011211	367397	429640
20011218	391995	456968
20011221	412581	471239

We will set up three individual arrays named cotDateArray, cotBullsArray, and cotBearsArray and assign corresponding data to each. Each element of each array will be linked by a single index variable. Element one in the cotDateArray will correspond with element one in the cotBullsArray and cotBearsArray arrays. Element one in the cotDateArray will not correspond with element two in the other arrays. To extract accurate data, you must span the arrays with the exact same index variable. You would access the first line of data in our table by using the following syntax:

```
myDate = cotDateArray[0];
myBulls = cotBullsArray[0];
myBears = cotBearsArray[0];
```

In the case of our arrays, the zero element in the cotDateArray is 20011016, in the cotBullsArray it is 378866, and in the cotBearsArray it is 415289. Can you see why we refer to these arrays as parallel? The next snippet of code shows how to declare, initialize and assign the three arrays.

```
Vars: barDelay(0),arrayIndex(0),bullPos(0),bearPos(0),iCount(0);
Arrays: cotDateArray[1000](0),cotBullsArray[1000](0),cotBearsArray[1000](0);

cotDateArray[ 0] = 19860115;cotBullsArray[ 0] = 29116; cotBearsArray[ 0] = 29119;
cotDateArray[ 1] = 19860131;cotBullsArray[ 1] = 31665; cotBearsArray[ 1] = 27329;
cotDateArray[ 2] = 19860214;cotBullsArray[ 2] = 37974; cotBearsArray[ 2] = 32357;
cotDateArray[ 3] = 19860228;cotBullsArray[ 3] = 42672; cotBearsArray[ 3] = 41513;
cotDateArray[ 4] = 19860314;cotBullsArray[ 4] = 44976; cotBearsArray[ 4] = 60318;
cotDateArray[ 5] = 19860331;cotBullsArray[ 5] = 31899; cotBearsArray[ 5] = 48075;
cotDateArray[ 6] = 19860415;cotBullsArray[ 6] = 29808; cotBearsArray[ 6] = 38246;
```

```
cotDateArray[ 7] = 19860430;cotBullsArray[ 7] = 28780; cotBearsArray[ 7] = 38111;
cotDateArray[ 8] = 19860515;cotBullsArray[ 8] = 52436; cotBearsArray[ 8] = 43043;
cotDateArray[ 9] = 19860530;cotBullsArray[ 9] = 53255; cotBearsArray[ 9] = 49183;
cotDateArray[ 10] = 19860613;cotBullsArray[ 10] = 55666; cotBearsArray[ 10] = 46562;
```

This type of programming is known as "grunt programming." It doesn't require eloquence, just redundant brute force. Concerning the *Commitment of Traders* report, we have done the work for you in a number of different markets. You can access the data from the enclosed CD-ROM and test for yourself the prophetic nature of the commercial interests. The first two letters of the file name represents the commodity symbol. You would then create a strategy and copy and paste the data into the EasyLanguage strategy. The following program demonstrates how to use the data in an actual trading strategy. We will skip over the arrays assignments.

```
if(barNumber = 1) then
begin
       value3 = cotDateArray[arrayIndex]-19000000;{Puts date into
          TradeStation format}
       value4 = Date;
       {If your bar chart starts after the first date of COT data then loop
          until we get pertinent COT data.}
       for iCount = 1 to 643
       begin
              value3 = cotDateArray[arrayIndex]-19000000;
              if(value3 < value4) then arrayIndex = arrayIndex + 1
       end;
end;
{At this point our bar chart should be either before or equal to the start of
the COT data}
value1 = cotDateArray[arrayIndex]-19000000;
value2 = Date;
if(value2[1] < value1 and value2 >= value1) then {Today's date equals a COT
   date}
begin
       {Start the delay—since the date of the COT report doesn't mark when
          the information is released for public consumption.}
       barDelay = 1;
       bearPos = cotBearsArray[arrayIndex];
       bullPos = cotBullsArray[arrayIndex];
       arrayIndex = arrayIndex + 1;
end;
if(value2 < value1 and barDelay = 3)then
begin
       if(bullPos > bearPos) then    {if the bulls = bears no new action}
       begin
```

```
            buy tomorrow at Open;
      end;
      if(bullPos < bearPos) then      {if the bears = bulls no new action}
      begin
            sell short tomorrow at Open;
      end;
end;
barDelay = barDelay + 1;
```

Again, this code may look daunting and confusing, but it really isn't. Basically, we are trying to align the dates of our bar chart with the dates of the COT report. If our bar chart starts after the first date of our COT data, then we must spin through the arrays until the COT date is equal to the start of our bar chart. If our bar chart starts before the start of our COT date, then we must spin through the bar chart until it is equal to the beginning of the COT data. The results from deriving our trades from the *Commitment of Traders* report are shown in Table 8.3.

Well, in the case of the S&P500, it looks like the commercial traders know what they are doing. Is this the case for other markets? We will let you figure that out. Again, the data is located on the companion CD-ROM.

Table 8.3
CotTrader Strategy Performance

Performance Summary: All Trades			
Total Net Profit	297,567.50	Open position P/L	93,525.00
Gross Profit	399,442.50	Gross Loss	(101,875.00)
Total # of trades	68	Percent profitable	64.71%
Number winning trades	44	Number losing trades	24
Largest winning trade	44,562.50	Largest losing trade	(17,375.00)
Average winning trade	9,078.24	Average losing trade	(4,244.79)
Ratio avg win/avg loss	2.13868	Avg trade (win & loss)	4,375.99
Max consec. Winners	6	Max consec. losers	4
Avg # bars in winners	73	Avg # bars in losers	18
Max intraday drawdown	(52,050.00)		
Profit Factor	3.92091	Max # contracts held	1
Account size required	52,050.00	Return on account	571.70%

DAY OF WEEK ANALYSIS

Many studies have been done on buying/selling on different days of the week to enhance profitability. I know of a number of market technicians that are firm believers in a day of the week for taking market action. A series of studies has been done with the objective of determining if buying or selling on different days of the week gives a trading edge. There are a number of different ways to look at this. Some are:

- Open to close on the same day
- Close to close from one day to the next
- Volatility of markets for different days of the week

One could introduce a variable stop for different days of the week depending on the volatility. This is a computer exercise to determine the best set of variables to produce the best possible return on past market action. This is known as curve-fitting or perhaps super curve-fitting. One must ask the question whether this will bear any relationship to future market action. It reminds us of the infomercial on late night television where computer studies were made on seasonality of all markets. Such studies would show that one buys say Swiss Franc on one day of the year and sells on another computer-optimized day for accuracies of 80 percent. Anyone who believes ideas such as this simply does not understand the forces of supply and demand. This is a computer-generated study that can produce any desired return one wants on past market data. The proponents of day of week trading say that people are more inclined to buy or sell on one day of the week versus another or their particular pattern works better. It has nothing to do with supply and demand characteristics. Is such an approach reliable or pure hogwash? We will let you decide, using the tables in the next section that show the results of our day of week analysis using EasyLanguage code to generate the test results.

Open to Close and Open to Open Relationships

Our first tests will analyze the Open to Close relationship on the various days of the week. The following code illustrates the code for buying or selling on the open on the different days of the week and liquidating the position on the close. We are using an Input statement so that we can test buys and sells across the different weekdays in one optimization run. This test will show if certain days have a bullish or bearish bias. We will test the Dow Jones, S&P500, NASDAQ, and U.S. Bonds.

```
Inputs: buyOrSell(1),whichDay(1);
{buyOrSell: 1 = buy 2 = sell whichDay : 1 is Monday 2 is Tuesday etc . . .}
if(DayOfWeek(Date of Tomorrow) = whichDay) then
begin
       if(buyOrSell = 1) then
              buy("Buy On Open") at next bar at Open;
       if(buyOrSell = 2) then
              sellShort("Sell On Open") at next bar at Open;
end;

SetExitOnClose;
```

You will have to set up an optimization run and vary the buyOrSell input from 1 to 2 and vary the whichDay input from 1 to 5 with increment values of 1. We are using TradeStation's optimization capabilities in a somewhat different manner than we have discussed thus far in the book. We really aren't trying to optimize a certain parameter. We simply want TradeStation to batch process the same program over different parameters. In other words, we aren't optimizing, we are batch processing. If buyOrSell is equal to 1, then the system will only buy. If this input is 2, then the system will only sell. The whichDay input determines which day the trade takes place. If whichDay is 1, then the trade takes place on Monday and so on. Table 8.4 shows the results of buying and selling on the various days of the week. We thought it would be of interest to break the test into two time periods: 1982 (or whenever the market started) through March 2000, and April 2000 through April 2002. The first test period covers the majority of a bull market and the second time period covers the current bear market. Tables 8.4 and 8.5 show the results of trading on the different weekdays over the two time periods.

Tables 8.6 and 8.7 show the results of almost the same test as Tables 8.4 and 8.5. The only thing different in this test is the positions were held overnight and liquidated on the next morning's open.

DAY OF WEEK VOLATILITY ANALYSIS

Is there a particular day of the week that shows more volatility on a consistent basis? Let's find out. This test was conducted by first measuring the 30-day volatility up to the week we were analyzing and then comparing the daily volatility against this measurement. We summed up the quotient of the true range of each weekday divided by the average true range of the past 30 days measured at the close of the previous Friday (TrueRange/AverageTrueRange(30)) and then divided by the number of occurrences of each weekday. Since we aren't analyzing profits or losses, we will rely on the Print Log to print out our findings. In this test, volatility is defined as the average true range for the past 30 days.

Table 8.4
Day of Week Analysis 1—Table 1

Day of Week Analysis Prior to April 2000	Buy/Sell on the Open and Liquidate on the Close							
	Dow Jones		NASDAQ		S&P500		U.S.Bonds	
Day of Week	Buy	Sell	Buy	Sell	Buy	Sell	Buy	Sell
Monday	$8,110	–$8,110	–$5,915	$5,915	$45,913	–$45,913	$1,625	–$1,625
Tuesday	–$18,130	$18,130	–$53,920	$53,920	$1,063	–$1,063	$50,031	–$50,031
Wednesday	$9,960	–$9,960	$36,140	–$36,140	$75,800	–$75,800	–$35,500	$35,500
Thursday	–$1,030	$1,030	$26,450	–$26,450	$29,613	–$29,613	$1,844	–$1,844
Friday	$4,920	–$4,920	$33,325	–$33,325	–$4,198	$4,198	$4,031	–$4,032
Total	$3,830	–$3,830	$36,080	–$36,080	$148,190	–$148,190	$22,031	–$22,032

Table 8.5
Day of Week Analysis 1—Table 2

Day of Week Analysis After March 2000	Buy/Sell on the Open and Liquidate on the Close							
	Dow Jones		NASDAQ		S&P500		U.S.Bonds	
Day of Week	Buy	Sell	Buy	Sell	Buy	Sell	Buy	Sell
Monday	$13,070	–$13,070	–$40,500	$40,500	$20,725	–$20,725	$7,063	–$7,063
Tuesday	$2,790	–$2,790	–$153,300	$153,300	–$32,950	$32,950	$7,969	–$7,969
Wednesday	–$1,830	$1,830	–$53,300	$53,300	–$36,612	$36,612	$2,781	–$2,781
Thursday	$4,690	–$4,690	$58,350	–$58,350	$22,650	–$22,650	$2,375	–$2,375
Friday	–$6,200	$6,200	–$85,700	$85,700	–$17,175	$17,175	–$8,781	$8,781
Total	$12,520	–$12,520	–$274,450	$274,450	–$43,362	$43,362	$11,406	–$11,407

Table 8.6
Day of Week Analysis 2—Table 1

Day of Week Analysis Prior to April 2000	Buy/Sell on the Open and Liquidate on the Close							
	Dow Jones		NASDAQ		S&P500		U.S.Bonds	
Day of Week	Buy	Sell	Buy	Sell	Buy	Sell	Buy	Sell
Monday	$8,110	–$8,110	–$5,915	$5,915	$45,913	–$45,913	$1,625	–$1,625
Tuesday	–$18,130	$18,130	–$53,920	$53,920	$1,063	–$1,063	$50,031	–$50,031
Wednesday	$9,960	–$9,960	$36,140	–$36,140	$75,800	–$75,800	–$35,500	$35,500
Thursday	–$1,030	$1,030	$26,450	–$26,450	$29,613	–$29,613	$1,844	–$1,844
Friday	$4,920	–$4,920	$33,325	–$33,325	–$4,198	$4,198	$4,031	–$4,032
Total	$3,830	–$3,830	$36,080	–$36,080	$148,190	–$148,190	$22,031	–$22,032

Table 8.7
Day of Week Analysis 2—Table 2

Day of Week Analysis After March 2000	Buy/Sell on the Open and Liquidate on the Close							
	Dow Jones		NASDAQ		S&P500		U.S.Bonds	
Day of Week	Buy	Sell	Buy	Sell	Buy	Sell	Buy	Sell
Monday	$3,180	–$3,180	–$6,600	$6,600	$17,300	–$17,300	$2,750	–$2,750
Tuesday	–$9,070	$9,070	–$194,050	$194,050	–$69,862	$69,862	$8,750	–$8,750
Wednesday	–$580	$580	–$23,850	$23,850	–$32,937	$32,937	$5,437	–$5,437
Thursday	–$2,480	$2,480	$36,900	–$36,900	–$13,650	$13,650	$2,781	–$2,781
Friday	–$4,100	$4,100	–$71,200	$71,200	–$11,125	$11,125	–$12,875	$12,875
Total	–$13,050	$13,050	–$258,800	$258,800	–$110,274	$110,274	$6,843	–$6,843

```
Vars: volMeasure(1),count(0),dayName("Monday");
Arrays: binArray[5](0),dayCntArray[5](0);
{binArray will keep track of the daily ATR ratios
binArray[0] = Monday
binArray[1] = Tuesday
binArray[2] = Wednesday
binArray[3] = Thursday
binArray[4] = Friday  → Remember, arrays are zero based in EasyLanguage}
if(DayOfWeek(Date) = Monday) then volMeasure = AvgTrueRange(30)[1];
{The DayOfWeek function returns 1-5 for Monday-Friday.
We subtract a 1 from whatever the function returns to make the arrays zero
based. If today is Monday, then DayOfWeek returns a 1. We then subtract 1
from that and use it as the index into our arrays. Mondays will be stored in
the zero element.}
binArray[DayOfWeek(Date)-1] = binArray[DayOfWeek(Date)-1] +
AvgTrueRange(1)/volMeasure;
dayCntArray[DayOfWeek(Date)-1] = dayCntArray[DayOfWeek(Date)-1] + 1;
if(lastBarOnChart) then
begin
       Print(SymbolName,Date:6:0,"Day of Week Volatility Study");
       for count = 0 to 4
       begin
              if(count = 0) then dayName = "Monday";
              if(count = 1) then dayName = "Tuesday";
              if(count = 2) then dayName = "Wednesday";
              if(count = 3) then dayName = "Thursday";
              if(count = 4) then dayName = "Friday";
              Print(dayName,binArray[count]/dayCntArray[count]);
       end;
end;
```

As you can see from the code, we aren't utilizing inputs and, therefore, aren't utilizing optimization. In this test, we don't need to inform the program on which day to take action or to buy or sell; we are monitoring all days simultaneously regardless of buying or selling. Table 8.8 shows the results from our testing.

All of these tests are interesting and the programming behind them may offer some educational benefit, but can we really use the information? If one really believes that this is a valid analysis, then one can curve-fit their strategy for different entries and exits on each day of the week depending on historical analysis. We guess our attitude is that you shouldn't fool yourself by buying such curve-fitting at face value. What we mean is, don't get carried away with any historic analysis and/or back testing. Overall, we personally find it difficult to draw firm conclusions from the tests. We do track systems that use this type of analysis in the *Futures Truth* magazine. In real-time, walk-forward testing, these systems have not demonstrated any better performance than systems that

Table 8.8
Day of Week Volatility Study

Day of Week Analysis Before April 2000	Daily True Ranges as a Percentage of 30-Day Avg. True Range			
	U.S. Bonds	NASDAQ	Dow Jones	S&P
Day of Week				
Monday	88%	101%	96%	97%
Tuesday	99%	118%	198%	108%
Wednesday	94%	105%	188%	101%
Thursday	99%	116%	239%	100%
Friday	124%	114%	374%	110%
Day of Week Analysis After March 2000	**Daily True Ranges as a Percentage of 30-Day Avg. True Range**			
	U.S. Bonds	**NASDAQ**	**Dow Jones**	**S&P**
Day of Week				
Monday	71%	84%	93%	89%
Tuesday	95%	95%	95%	95%
Wednesday	107%	259%	268%	129%
Thursday	110%	241%	298%	114%
Friday	119%	174%	365%	131%

do not use this form of analysis. Again, here are the numbers. Draw your own conclusions and we welcome your comments.

TIME OF DAY ANALYSIS

Time of day analysis is more utilized by day traders; if you are only executing trades on a monthly basis, then you really don't care what time of the day you entered the trade. Day traders enter and exit the market on the same day. Many of these traders feel that time plays an important part in the success of a trade. Many people would agree that waiting until after the first 30 minutes of trading on the stock indices will prevent whipsaws. The first 30 minutes of trading is usually heavily laden with news and consists of high volatility and indirection. Other traders believe that the true trend of the day is not established until after lunch. We have heard from many brokers and traders that early trading is for the amateurs and late trading is for the pros. We thought we would put these hypotheses to the test. In this time of day analysis, we tested entering the S&P500 futures markets at various times. The test starts out by buying the open and liquidating on the close. Each subsequent test waits an

additional 15 minutes before a trade is entered. We still exit market on close and do not incorporate any form of protective or profit objective stops. In addition to keeping track of profits or losses, we also keep track of the average excursion for each test. After we completed testing long trades, we then tested the short side of the market. We were able to accomplish this analysis by using TradeStation's optimization capabilities and the following programming code:

```
Inputs: buyOrSell(1),barDelay(0);
Vars: barCount(0),delayval(0),excursionVal(0),totalExcursionVal(0),
   trdCount(0), posString("Long");
{Test using 5-minute bars}

if(BarNumber > 1 and date <> date[1]) then begin
       if(excursionVal < 0) then totalExcursionVal = totalExcursionVal +
          excursionVal;
          trdCount = trdCount + 1;
end;
if(date <> date[1]) then barCount = 0;
barCount = barCount + 1;
if(barCount = barDelay or barDelay = 0) then
begin
       if(MarketPosition <> 1 and buyOrSell = 1) then
       begin
              buy next bar at open;
              excursionVal = 99999999;
              value1 = excursionVal;
       end;
       if(MarketPosition <> -1 and buyOrSell = 2) then
       begin
              SellShort next bar at open;
              excursionVal = 99999999;
              value1 = excursionVal;
       end;
end;

if(MarketPosition = 1 and BarsSinceEntry > 0) then
begin
       if(Low < EntryPrice) then
       begin
              value1 = Low - EntryPrice;
              excursionVal = MinList(value1,excursionVal);
       end;
end;
if(MarketPosition = -1 and BarsSinceEntry > 0) then
begin
       if(High > EntryPrice) then begin
              value1 = EntryPrice - High;
              excursionVal = MinList(value1,excursionVal);
```

```
        end;
end;
SetExitOnClose;
if(LastBarOnChart) then
begin
        if(buyOrSell = 1) then posString = "Long";
        if(buyOrSell = 2) then posString = "Short";
        print("*****************************************");
        print(Date:6:0,"Testing on entering a ",posString," after
        ",barDelay*5," minute delay");
        if(trdCount = 0) then trdCount = 1;
        print("On average the market goes against your position",
        totalExcursionVal/trdCount*BigPointValue);
        print("*****************************************");

end;
```

As you can see from the code, we relied on the Print Log to output the statistics of our analysis. We set up a batch process by varying the buyOrSell and barDelay input. We initially set the boundaries of the buyOrSell input to 1 and 2 with an increment of 1. The barDelay input boundaries were set to 0 and 24 with an increment of 3. Table 8.9 shows the results of our batch processing.

Table 8.9
Time of Day Batch Process Results

buyOrSell	barDelay	NetPrft	GrossP	GrossL	#Trds	%Prft
1	15	13100	311275	–298175	292	51
1	21	33825	292750	–258925	292	52
1	18	8175	289725	–281550	292	52
1	24	39075	292875	–253800	292	53
1	12	13650	321600	–307950	292	51
1	0	14000	371100	–357100	292	50
1	6	3175	340250	–337075	292	49
1	3	14650	358150	–343500	292	51
1	9	3775	329525	–325750	292	51
2	15	–13100	298175	–311275	292	48
2	21	–33825	258925	–292750	292	47
2	18	–8175	281550	–289725	292	47
2	24	–39075	253800	–292875	292	47
2	12	–13650	307950	–321600	292	48
2	0	–14000	357100	–371100	292	50
2	6	–3175	337075	–340250	292	51
2	3	–14650	343500	–358150	292	49
2	9	–3775	325750	–329525	292	48

When you initially view the Strategy Optimization report, it may be sorted by profitability. You can resort by clicking on any of the other column headings. Table 8.10 shows the contents of the Print Log.

These results may be the first step that leads you down the road of enlightenment. You may be able to use this information as a foundation to build a viable trading strategy. If not, at least you may be able to use the programming in your own research project.

Table 8.10
Maximum Excursion Printout

```
**************************************************
1020417 Testing on entering a Long after  0.00 minute delay
On average the market goes against your position -2477.06
**************************************************
**************************************************
1020417 Testing on entering a Long after  15.00 minute delay
On average the market goes against your position -2430.05
**************************************************
**************************************************
1020417 Testing on entering a Long after  30.00 minute delay
On average the market goes against your position -2374.32
**************************************************
**************************************************
1020417 Testing on entering a Long after  45.00 minute delay
On average the market goes against your position -2264.90
**************************************************
**************************************************
1020417 Testing on entering a Long after  60.00 minute delay
On average the market goes against your position -2161.47
**************************************************
**************************************************
1020417 Testing on entering a Long after  75.00 minute delay
On average the market goes against your position -2095.81
**************************************************
**************************************************
1020417 Testing on entering a Long after  90.00 minute delay
On average the market goes against your position -2042.47
**************************************************
**************************************************
1020417 Testing on entering a Long after  105.00 minute delay
On average the market goes against your position -1905.31
```

Table 8.10
(Continued)

```
**************************************************
**************************************************
1020417 Testing on entering a Long after  120.00 minute delay
On average the market goes against your position -1851.11
**************************************************
**************************************************
1020417 Testing on entering a Short after  0.00 minute delay
On average the market goes against your position -2330.39
**************************************************
**************************************************
1020417 Testing on entering a Short after  15.00 minute delay
On average the market goes against your position -2299.14
**************************************************
**************************************************
1020417 Testing on entering a Short after  30.00 minute delay
On average the market goes against your position -2176.45
**************************************************
**************************************************
1020417 Testing on entering a Short after  45.00 minute delay
On average the market goes against your position -2051.37
**************************************************
**************************************************
1020417 Testing on entering a Short after  60.00 minute delay
On average the market goes against your position -1992.81
**************************************************
**************************************************
1020417 Testing on entering a Short after  75.00 minute delay
On average the market goes against your position -1912.24
**************************************************
**************************************************
1020417 Testing on entering a Short after  90.00 minute delay
On average the market goes against your position -1797.86
**************************************************
**************************************************
1020417 Testing on entering a Short after  105.00 minute delay
On average the market goes against your position -1830.65
**************************************************
**************************************************
1020417 Testing on entering a Short after  120.00 minute delay
On average the market goes against your position -1817.04
**************************************************
```

PATTERN RECOGNITION

Recognizing patterns in bar graphs can be simple with the human eye. The computer, on the other hand, has a much harder time. This is a case where the human mind can apply fuzzy logic and a computer program can't. Fuzzy logic is logic that allows for imprecise and ambiguous answers to questions, and it is the basis for artificial intelligence. A double bottom pattern is simple to pick out from a chart, but try explaining the formation to someone who has never seen a bar chart without having a bar chart in front of them. You'll discover that you must be very precise with your explanation because a computer is like a child that must be instructed. We would explain the double bottom pattern in the following manner:

> The market makes a significant low pivot and then trades above that point. Eventually, the market makes another significant low pivot in the general area of the first low pivot. The market then rebounds and moves up for an extended period of time.

Figure 8.1 illustrates a double bottom in Microsoft.
Figure 8.2 illustrates a double bottom in ShowMe.

Our explanation of a double bottom probably sounds reasonable. A person with some experience with bar charts would probably be able to visualize the pattern in their mind. The problem is, we aren't describing a double bottom to a somewhat experienced technician. We need to describe it in terms

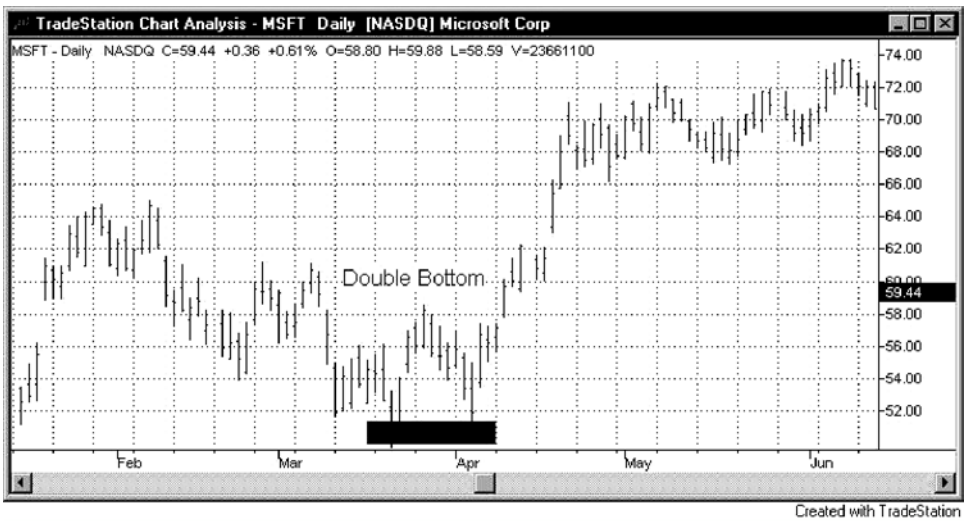

Figure 8.1 Double Bottom in Microsoft

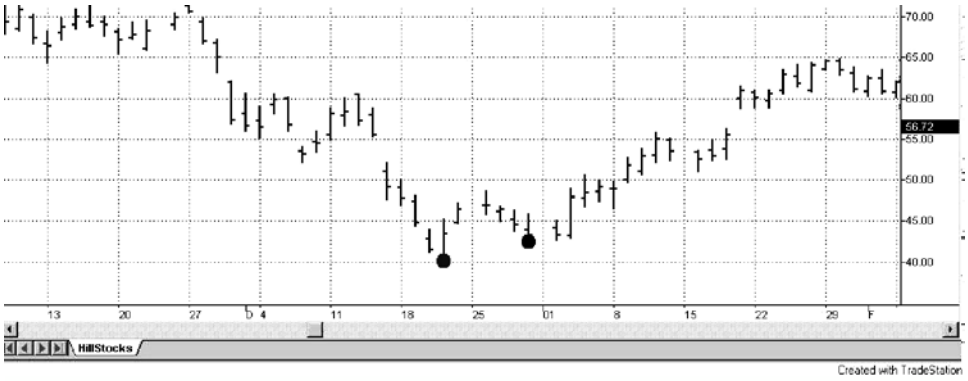

Figure 8.2 Double Bottom in ShowMe

that an inexperienced person or dumb computer would understand. The following is a description a computer might understand:

> The market makes a new twenty-day low and then forms a low pivot point. A low pivot point is a bar that has a lower low than the preceding and subsequent bar. The market then moves to a point that exceeds the highest high for the past five days and forms a high pivot point. A high pivot point is a bar that has a higher high than the preceding and subsequent bars. The market then moves back down and forms a low pivot point within one ten-day average true range of the first pivot low. The second pivot low can be above or below the first pivot low. You calculate a ten-day average true range by summing up the past ten day's true ranges and dividing by ten. The pattern is complete once the market has moved 3 ten-day average true ranges above the second pivot low. The pattern search is cancelled or reset if it is not completed within thirty trading days or the market makes a new twenty-day low without first making an intervening five-day high.

See how precise our description has become? Precision is great for programming, but not so great for pattern recognition. We have restricted our instructions in such a manner that there will be many double bottoms that we will miss. Unfortunately, we are stuck between a rock and a hard place. Again, this illustrates how much a computer lacks in intelligence and follows the instructions it is given to the tee. The following ShowMe code will pick out the pattern that we described. Hopefully, it will pick out something that looks similar to a double bottom.

```
Vars: state(0),tenDayATR(0),barCount(0),firstLowPivot(0);
Vars: secLowPivot(0),firstLowPivotPos(0),secLowPivotPos(0);

tenDayATR = AvgTrueRange(10);
```

```
if(state = 0 and Low[1] = Lowest(Low[1],20) and Low > Low[1]) then
begin          {initial state and we found a pivot low that is a 20-day low}
       state = 1;
       barCount = 0;
       firstLowPivot = Low[1];
       firstLowPivotPos = barNumber;
end;
if(state = 1) then
begin           {we are now searching for an intervening 5-day high pivot}
       barCount = barCount + 1;
       if(barCount > 30) then state = 0; {pattern not completed within 30
          bars}
       if(High[1] = Highest(High[1],5) and High < High[1]) then
       begin

              state = 2;
              barCount = barCount - 1; {subtract one bar—we will add one in
                 state 2}
       end;
       if(Low < firstLowPivot) then state = 0; {start over—lower pivot found}

end;
if(state = 2) then

begin         {now searching for the subsequent low pivot point}
       barcount = barCount + 1;
       if(Low < firstLowPivot - tenDayATR) then state = 0; {too far below
          first pivot}
       if(barCount > 30) then state = 0; {pattern not completed within 30
          bars}
       if(Low[2] > Low[1] and Low > Low[1] and
       (Low[1] < firstLowPivot + tenDayATR and
       Low[1] > firstLowPivot - tenDayATR)) then
       Begin {2nd low pivot price must be close to 1st low pivot price}
              state = 3;
              secLowPivot = Low[1];
              secLowPivotPos = barNumber;
              barCount = barCount - 1;
       end;

end;
if(state = 3) then

begin           {wait for market to move up and away from 2nd pivot low}

       barCount = barCount + 1;
       if(barCount > 30) then state = 0;
       if(High > secLowPivot + 2*tenDayAtr) then state = 4;
       if(Low < secLowPivot) then state = 0;
```

```
end;
if(state = 4) then
begin            {final state—if we made it here then we succeeded}

       Plot1[BarNumber - firstLowPivotPos + 1](firstLowPivot,"DoubleBottom");
       Plot2[BarNumber - secLowPivotPos + 1](secLowPivot,"DoubleBottom");
        state = 0; {Job is done—start over}

end;
```

This code utilizes an abstract programming construct known as a finite state machine. "What the $#%^%$# is a finite state machine and what does it have to do with trading systems?" you are probably asking. A *finite state machine* (or automata) is an abstract idea that can be used to pick predefined patterns out of a stream of data. These abstract machines sound complicated, but in actuality they are simple and easy to program; it's just the name that scares most people. These "machines" consists of a finite number of different states. States are conditions that have been met (you'll see when we explain the double bottom code). All finite state machines must have an initial and final state. The initial state for our double bottom pattern is the search for a new 20-day low. The final state occurs when the market moves three 10-day average ranges above the second pivot low. In addition to different states, these machines must also have a process in which one moves from one state to another.

Initially we set the state variable to 0 (our initial state). We then start searching for a pivot low that forms a new 20-day low. Once we find this point, we store the price of the pivot low and the bar number (remember, the bar number is the sequential numbering of the bars in a chart). We also start counting the number of bars from this point and set our state variable to 1. We have now moved from state 0 to state 1; the first criterion of the double bottom search has been satisfied. Now we search for a bar that makes a new five-day high and is a high pivot. If 30 bars elapse before we find a five-day high pivot or we experience a lower price than the first pivot low, then we start over. This is accomplished by setting the state variable back to 0. If we do indeed find a five-day high pivot within 30 bars, we then move on to state 2. State 2 is the search for a low pivot point that is close in price to the first pivot low. The two pivot points must be within one 10-day average true range of each other. Notice how we control the search criteria by the state variable. Again if 30 bars pass before we find the subsequent low pivot point, we bail out. Once we find an acceptable low pivot point, we store the pivot price, bar number, and move on to state 3. In state 3, we look for a retracement from the low pivot point. The market must move 3 average true ranges up from the low pivot point to satisfy our criteria. Again, if all of this is not completed within 30 bars, the whole process starts all over. If we do satisfy this last criterion, we move on to state 4 (our final state). In state 4, we use the Plot statement to draw a dot below the two low pivot points that form the double bottom:

```
Plot1[BarNumber - firstLowPivotPos](firstLowPivot,"DoubleBottom");
Plot2[BarNumber - secLowPivotPos](secLowPivot,"DoubleBottom");
```

Since there can be many bars between the pivot points, we keep track of the BarNumber of the pivot points so we can tell TradeStation where to draw the dot. Let's say the first pivot low occurred on BarNumber 25 and the current BarNumber is 62. You can instruct TradeStation to draw on the correct bar by subtracting the BarNumber that you would like to "point out" from the current BarNumber. In the case of our example, we would instruct TradeStation to draw 37 bars back: Plot1[BarNumber – firstLowPivotPos].

Any complicated pattern can be found through the use of finite state machines. The success of your search is based solely on how precise you program the search criteria. We have seen some pattern recognition search engines with as many as 15 different states. With the example of our double bottom finite state machine, you should now understand how to determine different states or steps and the processes to bail out of a search and move from one state to another.

INTERMARKET ANALYSIS

Have you ever noticed a correlation between two different markets? There have been many books written on this very subject. Our last research topic will discuss how to use EasyLanguage's multidata capabilities to test an idea based on the relationship between the U.S. Treasury bond futures market and the S&P500 futures market. These two markets compete against each other for investor dollars. If the stock market is bearish, many times investors will transfer funds from equities into bonds. If the stock market heats up, then the money will flow from bonds into the equities market. The test that we will perform will simply give you the basic tools to start your own intermarket research. This test will involve buying one S&P500 futures contract at yesterday's high when the eight day moving average of closing prices in the bond market crosses below the twenty-four day moving average. One contract of the S&P500 will be sold at yesterday's low when the eight day moving average of bond prices crosses above the twenty-four day moving average. Long positions are liquidated when the system enters a short position or the market penetrates a ten-day low. Short positions are liquidated when the system enters a long position or the market penetrates a ten-day high. Table 8.11 shows the performance of our intermarket trading system.

```
{System buys the S&P when the eight day moving average of the U.S. Bonds
crosses below the 24 day moving average of the U.S. Bonds. The opposite is
true for the sell side.}
if(Average(Close of data2,8) crosses below Average(Close of data2,24)) then
       buy("SPbuyUSdn") next bar at High stop;
if(Average(Close of data2,8) crosses above Average(Close of data2,24)) then
       sellShort("SPsellUSdn") next bar at Low stop;
```

Table 8.11
BOND-VS-SP Intermarket Results

TradeStation Strategy Performance Report—InterMarketSys @SP-Daily (5/3/1996-4/19/2002)

Performance Summary: All Trades

Total Net Profit	37,537.50	Open position P/L	(2,875.00)
Gross Profit	150,000.00	Gross Loss	(112,462.50)
Total # of trades	29	Percent profitable	37.93%
Number winning trades	11	Number losing trades	18
Largest winning trade	27,750.00	Largest losing trade	(20,575.00)
Average winning trade	13,636.36	Average losing trade	(6,247.92)
Ratio avg win/avg loss	2.1825	Avg trade (win & loss)	1,294.40
Max consec. Winners	4	Max consec. losers	7
Avg # bars in winners	26	Avg # bars in losers	6
Max intraday drawdown	(56,062.50)		
Profit Factor	1.3338	Max # contracts held	1
Account size required	56,062.50	Return on account	66.96%

```
if(MarketPosition = 1) then sell next bar at Lowest(Low,10) stop;
if(MarketPosition = -1) then buyToCover next bar at Highest(High,10) stop;
```

CONCLUSIONS

The research capabilities of EasyLanguage and TradeStation were discussed as well as the usage of external data, day of week and time of day analysis, pattern recognition, and intermarket analysis. We can use TradeStation's optimization tool to perform batch processing. With TradeStation and its huge library of data and the concepts that we discussed in this chapter, the ideas that you can research are unlimited. All good trading systems start out as well-researched ideas.

This ends our instruction on programming EasyLanguage. We hope the ideas and programming techniques that we presented will be of great use to you. You will soon discover that the development and discovery of good, sound market principles will be much more difficult than programming them. The rest of the book is dedicated to the utilization of percent change charts, a beginner's guide to options, and interviews with some of today's leading system traders and vendors. Good luck and good system development!

9

Using TradeStation's Percent Change Charts to Track Relative Performance*

We ran into Jan Arps, and because of his knowledge and experience with TradeStation products, asked if he would like to contribute to our book. He accepted our offer and wrote an interesting chapter on selecting markets through the use of TradeStations' Percentage Change charts.

Technical analysis generally consists of two parts: *selection* and *timing*. Selection is the process of deciding which stock, mutual fund, option or futures contract, among the thousands available to you, to trade at any particular point in time. Timing, on the other hand, is the process of determining *when* to enter and exit a trade once you have selected a suitable tradeable.

There are numerous methods, both fundamental and technical, that can help in the selection process. One of the selection questions many traders ask is, "How has this stock performed relative to alternative choices recently, and which is likely to be the strongest in the upcoming time frame?" In order to answer this question, we must be able to provide a level playing field for comparing tradeables whose various prices could range from a dollar to hundreds of dollars.

For example, let's say you are comparing a $10 stock with a $100 stock. Since the beginning of the quarter, both of these stocks have gone up exactly $10. If you plot these stocks on the same standard price chart with a linear price scale, the charts of the two stocks will show an identical increase for the quar-

*Jan Arps, President of Jan Arps' Traders' Toolbox, is one of the leading authorities on TradeStation today. He has been trading and developing trading tools since 1952. Jan and his catalog of 400 TradeStation add-ins can be found at www.janarps.com.

ter. In fact, however, the $10 stock has doubled in price, while the $100 stock has only increased by 10 percent.

What we really need is a "normalized" way to display the behavior of multiple tradeables on the same chart so that we can compare their percentage change behavior over a specific time period. TradeStation's Percent Change chart feature gives you the ability to plot the change in price of any number of tradeables on the same chart, as a percentage of their price beginning at a specified starting date.

All Percent Change charts require that you specify an anchor bar. This is the bar from which all percent change calculations are made. The anchor bar can be the first bar of the chart, the last bar of the chart, or any bar that you may select between the first and last bars. Once an anchor bar has been selected, a percent change value is calculated by TradeStation for each bar on the chart by subtracting the average price of each bar from the closing price of the anchor bar and dividing this value by the closing price of the anchor bar.

For example, if the price of a stock at the anchor bar was $100 and the price ten days ago was $90, then the relative price on the percent change chart for the bar ten days before the anchor bar will be displayed as –10%. If the price 30 days after the anchor bar is $125, then the percent change chart will display +25% on that bar.

The procedure for creating a Percent Change chart in TradeStation is:

1. Right-click anywhere on a price chart, or click on *Format* in the menu bar.
2. Click on *Percent Change Chart*.
3. Click on *Enable*.
4. Select your anchor bar by clicking on either *Calculate from first bar*, *Calculate from last bar*, or *Calculate from this bar*.
5. To select your anchor bar when using *Calculate from this bar*, place the cursor on the selected anchor bar on your chart, right click and select *Calculate from this bar*. The chart will then show the price intersecting the zero line at your selected anchor bar. All other prices before and after the anchor bar will be displayed as the percentage change in price relative to the anchor bar.
6. To display a vertical line showing the location of the anchor bar on your chart, click on *Format Window* and click on the box labeled, *Show Calculation Point Marker*.

When you look at a collection of stocks on a Percent Change chart, it tends to move up and down more or less together, responding to overall market fluctuations. However, stocks with increasing strength will begin pulling out of the pack and you will see these stronger stocks crossing above the less-strong stocks. Conversely, weakening stocks will begin crossing below the lines of

stronger stocks. This is one of the main characteristics we look for on a Percent Change chart. As buyers, we are looking for stocks pulling out of the pack. As sellers, we are looking for weakening stocks dropping down through the pack.

It's sort of like a horse race, with horses moving up and falling back within the pack as the race progresses. The difference between selecting a winning horse in a horse race and selecting a winning stock, however, is that you can't change your bet in the middle of a horse race. In the stock market there is no finish line, and you can switch from a struggling stock to a stronger stock as often as you want, at any time in the "race."

WORKING WITH PERCENT CHANGE CHARTS

So how do we use Percent Change charts to identify fast-moving stocks? As an example, let's suppose we have chosen four different companies to follow:

1. Amalgamated Aquanautics Corp.
2. Better Biscuits, Inc.
3. Cheerful Car Rental Corp.
4. Destiny Drugs & Chemicals, Inc.

Figure 9.1 shows a chart of each of these four companies, plotted in the same window with the same linear price scale.

Figure 9.2 shows a Percent Change chart of these same four companies from December, 2000 to June, 2001. This chart has been created using January 1, 2001 as the anchor bar. We see from this chart that by June, stock 1,

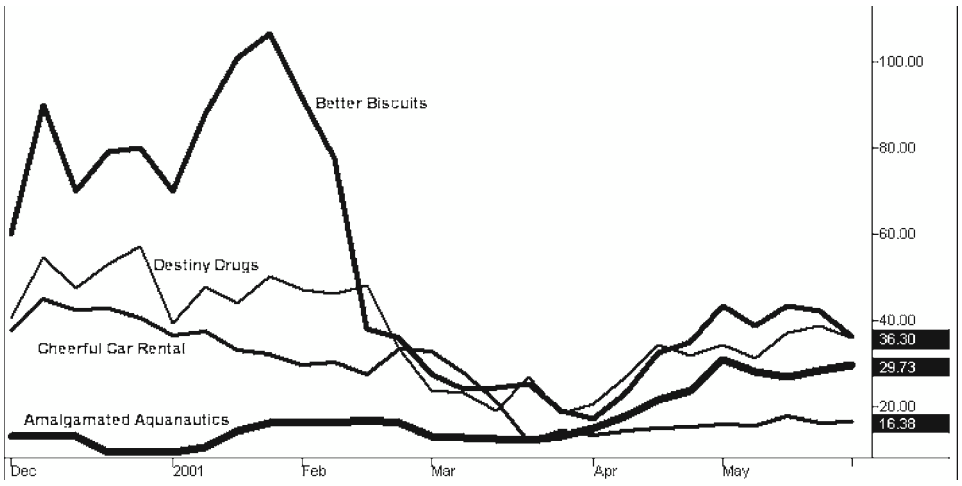

Figure 9.1 Four Stocks Plotted on Same Price Scale

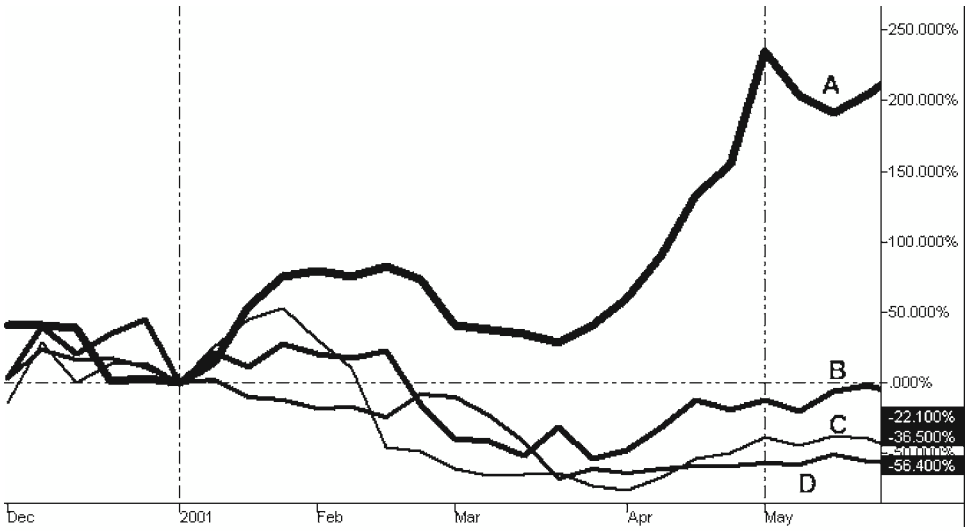

Figure 9.2 Four Stocks Plotted on a Percent Change Chart

Amalgamated Aquanautics, was up over 200 percent over the period; Stock 2, Better Biscuits, was essentially flat for the year; stock 3, Cheerful Car Rental, was down 36 percent; and stock 4, Destiny Drugs, was down 56 percent.

We can conclude from examining the chart in Figure 9.2 that we would have achieved the greatest return between January and June, 2001 had we bought Amalgamated Aquanautics on January 1. But the question we *really* want an answer to is, "How would we have known to choose Amalgamated back in January?"

A momentum trader will say, "choose the strongest stock and go with it until it is surpassed by one of the others. The power of the Percent Change chart is its ability to display the relative change in prices of a number of stocks through a given time period. During the time shown in the chart in Figure 9.2, the stocks changed places a number of times in their relative rate of change. Although Amalgamated Aquanautics ended up having the greatest percentage increase in value since January, it had its biggest upward move between March and May. Examining the chart more closely, we see that Cheerful Car Rental initially was quite strong as well, but by mid-February it was dropping down through the pack.

Figure 9.3 is the same chart as Figure 9.2, but with the anchor bar set at the end of the chart, on June, 2001. Notice how different the two charts look. You see that the strongest stock, Amalgamated Aquanautics, rose from below zero to the zero line in Figure 9.3, while the weakest stock, Destiny Drugs, dropped down from above the zero line. This is an important fact about percent change charts. The strongest stocks prior to the anchor bar will appear to

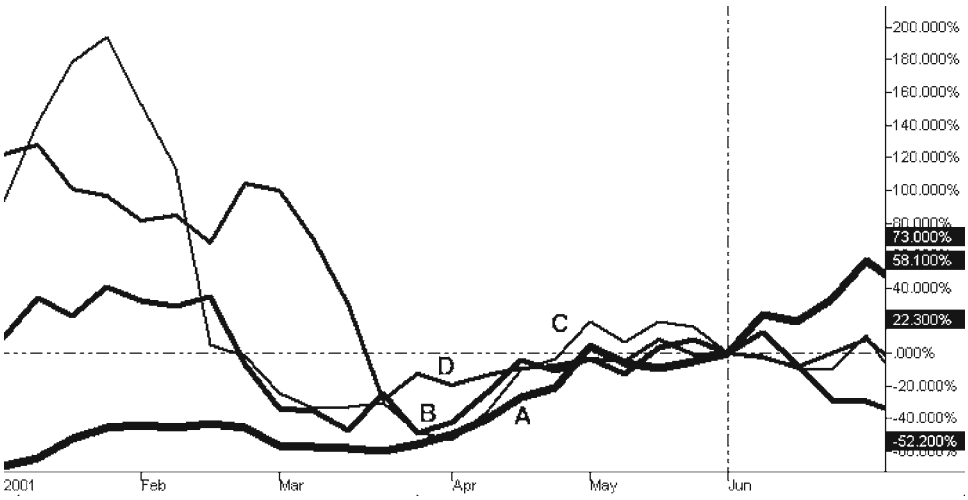

Figure 9.3 Chart of Four Companies with an Anchor Bar

be rising from the bottom, while the weakest stocks will appear to be falling from the top. You need to recognize this characteristic when using Percent Change charts using past history to identify a strongly advancing or declining stock.

Let's follow the progress of our four stocks month by month during the six-month sample period, moving our anchor bar forward one month at a time so that we can see what the chart looks like to an observer at the end of each month. The anchor bar is at February 1 in Figure 9.4. We see that between

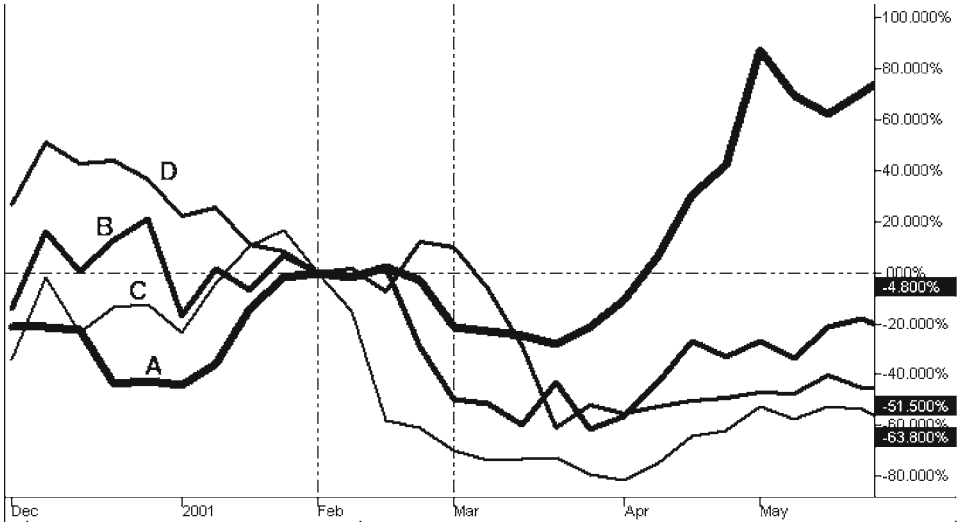

Figure 9.4 Anchor Bar Moved to February 1

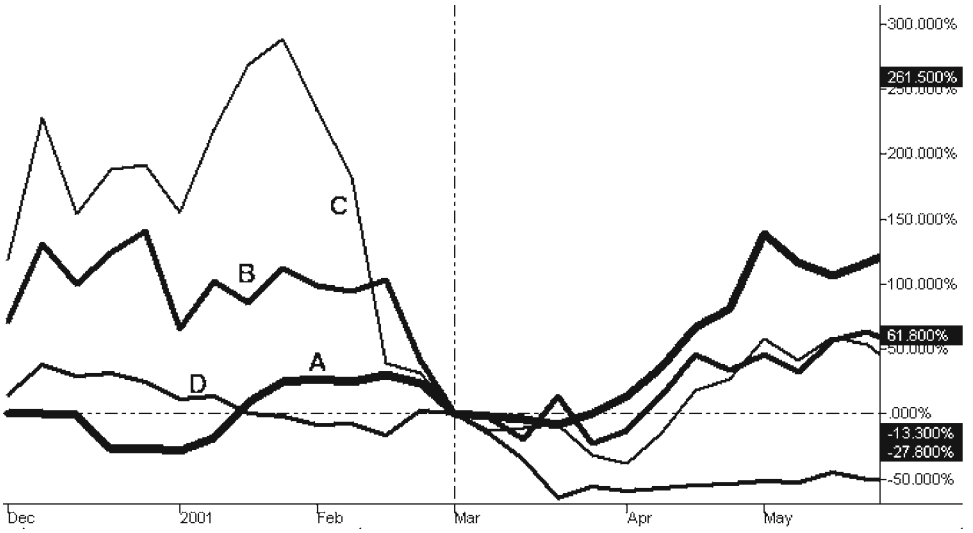

Figure 9.5 Anchor Bar Moved to March 1

January 1 and February 1, Amalgamated Aquanautics has begun to move up sharply, Cheerful Car Rental has also moved up, while Better Biscuits has remained relatively steady, and Destiny Drugs is weakening. Amalgamated Aquanautics is clearly the strongest stock at this point, and should be our prime buying candidate.

In Figure 9.5, the anchor bar has been moved to March 1. During the month of February, Cheerful Car Rental has continued its downward trend, along with Better Biscuits. Destiny Drugs rose marginally and Amalgamated Aquanautics experienced a pullback from its strong showing in January. At this point no new leader has emerged from the pack that would encourage us to change our pick from Amalgamated.

In Figure 9.6, the anchor bar has moved to April 1. During the month of March, Amalgamated is once more moving up from below, Destiny and Cheerful continue to weaken, and Better Biscuits is holding its own. Conclusion: Stick with Amalgamated.

In Figure 9.7, the anchor bar has moved to May 1. During the month of April, all four of our stocks rose. Amalgamated continued to be the strongest. Cheerful made a dramatic reversal, more than Amalgamated. Better Biscuits improved significantly, and Destiny moved up marginally. At this point, we are beginning to see a change in leadership. With Cheerful coming up through the pack past Amalgamated, we should consider adding Cheerful to our portfolio.

Looking back at June 1 in Figure 9.2, Amalgamated Aquanautics turned out to have been an excellent pick in February, having grown by almost 200 percent between February and June.

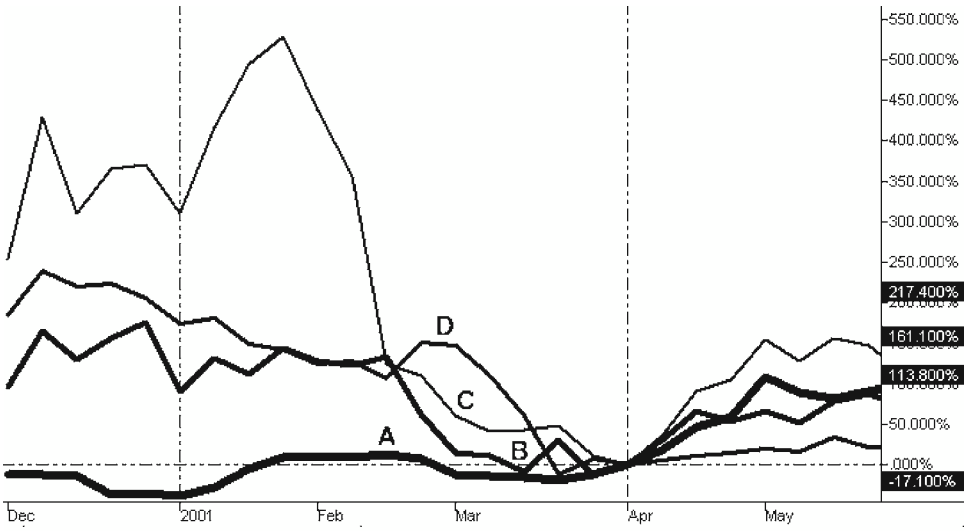

Figure 9.6 Anchor Bar Moved to April 1

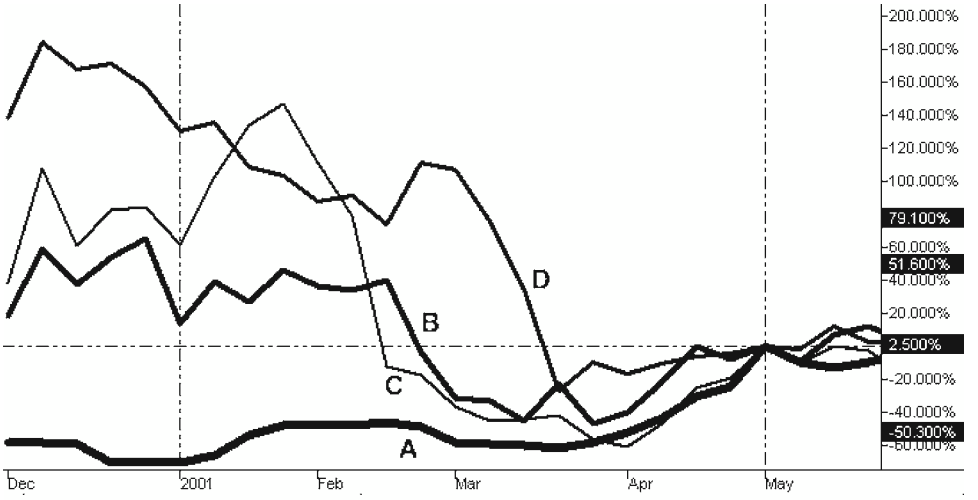

Figure 9.7 Anchor Bar Moved to May 1

CONCLUSIONS

Percent Change charts are a useful tool in providing a level playing field for comparing the performance of a group of stocks or commodities in order to select the most likely prospect for future growth.

10

Options*

We wanted to include a chapter on Options because they are yet another weapon that can be included in one's trading arsenal. Options are confusing to most traders and, therefore, are overlooked. Len Yates, with his expertise, clears up the confusion and introduces several option strategies. The ideas that are presented can be used with most option analysis software.

Being from the Chicago area, host to some of the world's largest options exchanges, I know or encounter many people whose work is related to options trading. So I have the advantage of being surrounded by plenty of locals who are familiar with options and how they work. Still, there are likely even more people *un*familiar with options.

During the course of many social engagements, I invariably end up leading a discussion on options and what options are. Without even finishing my standard overview, I am invariably met with a response such as, "Well, that sounds too complicated for me" from a befuddled listener. This usually leads me to the snack table because I don't want to make any party listen to what might be to them an arcane subject.

But what I want to say, and sometimes do, is that options are not really that complicated if you're committed to learning the terminology and the basic principles of the game. I like comparing options to chess: if you spend an hour learning the rules, you may find you like the game and can win at it, too.

---

*This chapter was written by Len Yates, a leading option authority and software developer. He is president of OptionVue Systems International, Inc. He can be reached at www.optionvue.com.

Of course, practice is key to becoming successful. Options have a number of strategies with which you need to be familiar, but hardly as many as in chess! However, anyone with average intelligence can learn all about options in a relatively short span of time.

As most brokerage firms allow you to trade stocks they also allow you to trade stock or index options, and it's pretty easy to set up an account and start trading. Also, almost every brokerage firm that allows you to trade futures also allows you to trade futures-based options. Establishing an options trading account requires a little extra paperwork, including a statement that you have read and understood the options prospectus and are prepared to assume the risks involved.

As you progress in your education, you'll start to understand that the unique qualities of options make them fascinating to trade. They can be used in a number of strategies, including using two, three, or more options in combination, and using one or more options in conjunction with a position in the underlying to deliver risk/reward characteristics that cannot be matched by simply buying or selling the underlying.

Options trading, though, is not for everyone. There are some conservative options trading strategies, and there are some risky strategies as well, where your capital can be lost very quickly. It is up to you, after learning as much as you can before making a single options trade, to decide if you have the right temperament for it.

OPTION BASICS

Basic option principles start with understanding the agreement between the buyer and seller. Let's suppose you agree to sell your car to another party. As part of your agreement, you and the other party have agreed on a sale price of the car and a time for it to be delivered to the buyer. This type of agreement is called a *forward contract*. Now, if you only agree to let someone retain the *right* to buy the car from you for a stated price and only for a limited time, you have sold an *option*. Therefore, the holder of the option possesses the right, but not an obligation, to buy something at a stated price for a limited time. So the party who sold the option is obligated to deliver the car if the option holder decides to exercise his right, or his option, to purchase the automobile.

Any asset designated to be delivered in such an agreement (the car in this example), is called the *underlying asset*, or *the underlying* for short. The price both parties agreed to for buying and selling the car is called the *strike price* of the option.

Let's say my car, an antique collectible, is worth $100,000. I could agree to let someone have an option to buy the auto from me for, say, exactly $100,000 anytime during the next two years. The option's strike price is there-

fore $100,000, and the underlying is the car itself. Now, why would I enter into such an agreement? After all, if the car appreciates over the next two years, that appreciation would be lost to me because I have agreed to sell the car for $100,000. Furthermore, I am locked into owning the car, and may not sell it to anyone else for the next two years—because if the option holder decides to exercise his right, I am obligated to deliver the car. So, why should I put myself in such a constrained position?

First, for the money I receive. An option has value and won't be granted without some form of payment. For this particular option, I may require $15,000. The $15,000 (should the buyer agree to purchase the option at this price) would be mine to keep regardless of whether the option holder later decides to exercise his right to purchase the car.

Second, I may be unwilling or unable to sell the car at this time. I might be happy to receive the $15,000, especially if I believe that the car will not appreciate $15,000 over the next two years. If the car appreciates less than $15,000, I'm better off for having sold the option. If the car appreciates exactly $15,000 during the next two years, there's no net gain as I end up with the same amount as if I had not sold the option. And if the car appreciates more than $15,000, then I may regret having sold the option.

Why might someone want to *buy* an option in the first place? For starters, leverage. For only $15,000, the option buyer can have control over a $100,000 asset. Without incurring the hassle of ownership, he has the *right* to own the car anytime simply by submitting an exercise notice and paying the agreed $100,000. Suppose, during the next two years, he changes hobbies, and no longer collects cars but beer cans instead. He now has greater flexibility of getting out of the deal because, in fact, he never got in; he never bought the car. He simply lets his option expire. Also, he may believe that the car will appreciate more than $15,000 during the next couple of years, presenting the possibility of simply exercising his option and then selling the car for more than $115,000.

Another reason to buy an option, rather than the asset itself, is the limited risk. Collectibles don't often drop in price, and if the value of his car, for whatever reason, were to fall below $100,000, the option holder is not likely to exercise. (Why should he pay $100,000 for something that could be bought, on the open market, for less than $100,000?) And if the value of the car were to fall to less than $85,000, the option buyer would be happy that his loss is limited to the $15,000 he paid for the option, rather than having bought the car and now seeing a loss of more than $15,000.

Does an option have to have a strike price precisely equal to the car's current fair value? Of course not. I might rather have written (sold) my option at a strike price of, say, $110,000—$10,000 above the current fair value. Such an option wouldn't be worth as much, however, and I probably would not be able to get $15,000 for it. When an option's strike price is above the underlying

Table 10.1
An Option Is Exercised

If the option holder exercises, the option holder:	The option writer is assigned, the option writer:
• Pays for the asset.	• Receives payment for the asset.
• Receives the asset.	• Must deliver the asset.

Table 10.2
An Option Expires

If the option holder lets the option expire, the option holder:	The option writer gets to:
• Does nothing.	• Keep his asset.
• Loses premium paid.	• Keeps premium paid.
No additional cash changes hands.	

asset's current market value, the option is said to be *out of the money* (discussed later). I might prefer selling an out-of-the-money option because it gives my asset room to appreciate.

Does this option have to end either in exercise or by letting it expire? No, there is a third possible outcome. If the two parties are willing, and can agree on a price, the option seller may buy back his option, effectively canceling it out. Tables 10.1 and 10.2 summarize the transaction and terms involved. When an option is transacted, the buyer (holder) pays the seller (writer) the agreed amount (premium) for the option. This premium is the writer's to keep, no matter what.

LISTED OPTIONS

In our example, an option was transacted between two individuals. Its strike price (price of the asset), premium (cost of the option), and duration (expiration date) was created by agreement and negotiated by the two parties to meet both of their needs. In contrast, there are the *listed options* traded on the public exchanges on many stocks, indexes, bond futures, commodity futures, and currency futures. There are even listed options on interest rates, inflation rates, and the weather.

These listed options have standardized features that appeal to a large group of traders and help to build a liquid market. For starters, several strike prices are usually available at regular price intervals. And several different dura-

tions (expiration dates), following a set pattern, are usually available. In stocks, for example, one set of options expires in 30 days or less, another set expires in approximately 31 to 60 days, another set in approximately 3 to 6 months, and so on, going out as far as 2 years or more.

Each listed option is also for the same quantity of the underlying asset. In stocks, one option is based on 100 shares of the underlying stock. In futures, one option is based on one futures contract.

As the markets are constantly moving, options prices are continuously quoted and changing. This is possible because market makers at the options exchanges are always publicly posting prices at which they are willing to buy and sell (bid and asked prices). They stand ready to take the other side of your trade, and thus "make a market" in the options they are responsible for. So an option holder may sell his option(s) at any time, and an option writer may buy (to close his option position) at any time.

If an option holder exercises his option, the Options Clearing Corporation assigns any party holding a short position on a random, arbitrary basis. So an option buyer never finds out, nor should he care, who sold the option to him. And the option seller never finds out, nor does he care, who bought the option from him.

In our car example, it is possible, even likely, that the option holder will exercise his option prior to expiration. It's more of a direct personal agreement. In contrast, the vast majority of listed option buyers never exercise; they simply sell their options on the open market. Many of these people are speculators who only expect to hold their option for a short time. Once the underlying makes a move in the expected direction (or perhaps a move in the wrong direction), they sell. In a sense, up to expiration day, options are like hot potatoes being tossed around among the market of speculators. Such activity accounts for quite a bit of the options trading volume but not all of it. Another major source of trading volume is institutional investors. They may use options to hedge large positions, or simply trade large positions for speculation.

Until now we have only talked about options to *buy*. Options to buy something at a stated price for a limited time are *call options*. But there is another type of option—an option to *sell* something. While these options can be a bit more difficult to conceptualize, options to sell something at a stated price for a limited time are known as *put options*.

Nomenclature and Terminology

An option is specified by stating its underlying asset, the expiration month, the strike price, and the type (call or put), usually in that order. For example:

IBM July 120 calls.

This would usually refer to options expiring in July of the present year. If the options expire more than a year out from the present day, one might need to include a year indication of some kind, for example:

IBM July03 20 calls ("03" meaning the year 2003)

Also important is the manner options prices are converted into dollar amounts. Most stock and index options have a multiplier of 100, meaning that one option is for 100 shares of stock. So, if you were to buy one option at a price of 1.10, for example, you would pay $110. Multipliers for futures-based options vary from 50 to 500.

Long and Short

Most investors understand the concept of being *long*, whether they realize it or not. When you own something you are said to have a long position in it. As such, when you are long the market, you are taking a position to profit in a rising market.

Going *short* means to sell something, without first owning it, to profit from a falling market. How can you sell something you don't own? In securities trading, going short involves borrowing the securities (usually from your broker) to sell. When you move to close the position at a later date, you buy back the securities, giving the shares back to your broker. With futures and options, it's even easier. You're entering into an agreement that comes complete with standard contractual rights and obligations. The only difference is that futures and options afford you the opportunity to get out of the contract at any time by placing a buy or sell order that cancels your position before the obligations come due.

Being long or short an option does become more complex because of the two types of options: calls and puts. When you buy an option, regardless of whether it's a call or put, you are long the option. When you buy a call option, because you stand to benefit from the underlying going up, your position can be considered to be, in a general sense, long the underlying as well. However, when you buy a put option, because you stand to benefit from the underlying going down, you can be considered to be, in a general sense, short the underlying. Table 10.3 summarizes the four possible scenarios.

It is important that the options trader be familiar with the following terms and concepts. Note that our examples refer to stock options. However, the same terms and concepts apply to all asset types.

The value of an option is comprised of two components—*intrinsic value* and *time value*. As these two components are never quoted separately, all you see is the total price of the option. Nevertheless, understanding these two values and how they impact an option's total value is important.

Table 10.3
The Four Possible Scenarios of Buying/Writing Calls/Puts

Position	Exposure
Long calls	Long the underlying
Short calls	Short the underlying
Long puts	Short the underlying
Short puts	Long the underlying

To draw an analogy, the value of a company can be said to consist of (1) book value, plus (2) all the rest. *Book value*, meaning the company's value if one were to break it up and sell all of its assets, is similar to an option's intrinsic value. *All the rest*, including intangible assets and earnings potential, is like an option's time value.

Intrinsic value is what you could gain by exercising the option and immediately closing your new position in the underlying. For example, say the price of AOL is 30 and you hold a 25 call. You know that if you were to exercise your option, you'd pay 25 a share for the stock. After selling it on the market for 30, you'd realize a profit of 5 per share on the stock itself. Thus, the intrinsic value of the option is 5. Intrinsic value can also be considered the money component of the option's value. For instance, an option is said to be *in* the money when it possesses some intrinsic value. Call options are in the money when their strike price is below the current price of the underlying (as in the example above). It's the reverse for put options, which are in the money when their strike price is above the current price of the underlying.

When an option's strike price is *equal to* (in practice, very close to, as the market is constantly changing) the price of the underlying, it is said to be *at* the money.

Call options are said to be *out of* the money when their strike price is above the current price of the underlying. Put options are out of the money when their strike is below the current price of the underlying.

Time value, the other component that comprises an option's value, accounts for the potential of the underlying moving in the option's favor (up for calls; down for puts) during the remaining life of the option. Of course, it's just as possible for the stock to move against the option. However, if the stock moves the wrong way, an option's value can drop, at most, to zero. On the other hand, if the stock moves in the direction of your position, the option's value can theoretically go up without limit. And that's why options almost always retain some time value up to expiration.

Time value is the summary of all the possible intrinsic values the option might have at all of the possible underlying prices, on or before expiration, taking into consideration the probabilities of the stock reaching each of those

prices. As you may imagine, time value can be a challenge to estimate in your head or on paper. For that reason, options traders refer to mathematical models, implemented in computer programs, to compute the fair value of an option.

In the previous AOL example, we showed how an in-the-money option could be exercised to get into a stock position at below market price. Now when would it make sense to exercise an out-of-the-money option? Never. To exercise an out-of-the-money call would be to pay more than the current market price for a stock. To exercise an out-of-the-money put would be to sell a stock for less than the current market price of the stock. Unless you like throwing money away, it never makes sense to exercise an out-of-the-money option.

In fact, it rarely makes sense to exercise an in-the-money option either. Why? Because you'd be throwing away its time value. Let's go back to the AOL example. You have a 25 call and the stock is currently at 30. Your call, if it has more than a few days of life left, is probably worth something more than 5; say 6.5 (this would be an intrinsic value of 5 plus a time value of 1.5). If you exercise the option and then sell the stock, as before, you gain $500 on the stock transaction. However, your option was worth $650. So, you just lost $150—the option's time value. It would be better to sell your option on one of the options exchanges. You would receive the full value of your option of $650, and you would only need to perform (and pay for) one option transaction versus two transactions the other way.

Let's "pop" a little quiz to see if you have grasped the concepts we have been discussing.

1. A stock is at 65. A 70 call on this stock has a price of 1.85. Is this option in-the-money, at-the-money, or out-of-the-money? What is this option's intrinsic value? What is this option's time value?

   Answer: The option is out-of-the-money, has an intrinsic value of zero, and a time value of 1.85.

2. A stock is at 80. A 70 call on this stock has a price of 11.60. Is this option in-the-money, at-the-money, or out-of-the-money? What is this option's intrinsic value? What is this option's time value?

   Answer: The option is in-the-money, has an intrinsic value of 10, and a time value of 1.60.

3. A stock is at 40. A 45 put on this stock has a price of 6.30. Is this option in-the-money, at-the-money, or out-of-the-money? What is this option's intrinsic value? What is this option's time value?

   Answer: The option is in-the-money, has an intrinsic value of 5, and a time value of 1.30.

4. A stock is at 55. A 55 call on this stock has a price of 3.40. Is this option in-the-money, at-the-money, or out-of-the-money? What is this option's intrinsic value? What is this option's time value?

Answer: The option is at-the-money, has an intrinsic value of zero, and a time value of 3.40.

5. One further question: In question #2, do you think this option could trade below 10 (its intrinsic value)?

   Answer: Absolutely. In an open market, anything could happen. However, from a practical point of view, it would not trade for much less than 10. Traders are constantly prowling the options markets for bargains. When they see an undervalued option, they buy it in an instant. So, if you were to offer this option for sale at 9.8, someone would quickly buy it because they know they can immediately exercise it and sell the stock, realizing a 0.2 profit.

Remember, even if an option's time value has dropped to zero, it still retains its intrinsic value. You should be able to sell it for that, or perhaps just a bit less. This is said to be trading *at parity*.

Previously, I pointed out that many options are never exercised. It does not make sense to exercise an option that has any appreciable time value; you would be throwing away money. However, when an option's time value is zero or nearly zero, option holders *are* likely to exercise. Conversely, if you sell (short) an option with zero or nearly zero time value, you are apt to be assigned—and it can happen that very day. Early assignment may or may not be a significant danger to you. It depends on the nature of the position you would be left holding.

Closing Option Trades

The best way to close an option position prior to expiration is to execute an opposing transaction in the market. This is true whether you are holding puts and calls, and whether you are long or short. If you previously bought a call, the only way to close your position is to sell the same call. Some may think buying a put will do, but it will not. Also, selling another call on the same underlying does not do it. In fact, such trades might reduce your risk, but they would only build (and complicate) your original position.

The two alternatives to closing a position with an opposing transaction is to let the option expire or to exercise it. If your option remains out-of-the-money at expiration, it has no value and should be allowed to expire. However, if the option is in-the-money at expiration, it has value and should be exercised.

When you exercise a stock option, you pay for and receive shares of stock. When you exercise futures options, you are immediately in a futures position and no cash changes hands. When you exercise index options, you simply receive the intrinsic value as a cash settlement. There is no delivery besides cash, as exercising index options does not create a new position in another

security. If you have an index option that is 3.00 in-the-money at expiration, $300 is credited to your account.

The option holder isn't always required to submit an exercise notice. For example, exercise is automatic for some instruments when the option is a certain amount in-the-money. It is important to understand what will happen if you do nothing with an in-the-money option at expiration. Confer with your broker if you are unsure. It doesn't hurt to submit an exercise notice, as you never want to let a valuable option disappear!

American Versus European Options

Options can also be classified in terms of style, which relates to the two ways in which they can be exercised. If an option can be exercised any time up until expiration, it is said to be *American style*. If an option can only be exercised on expiration day, it is said to be *European style*.

These references are not concerned with the continent on which the options trade. American and European style options trade in America, Europe, and elsewhere around the globe. For instance, in the United States all stock options and over half of the index options are American style (the remaining are European style). Some futures-based options are American style and others are European style.

An option buyer intending to exercise must understand which style he is purchasing. An option seller might prefer European style options, because he might not want to be concerned about assignment before expiration. For the options buyer who has no intention of exercising, the only difference is that American style options are a bit more valuable—and a quality options pricing model will bear this out.

The Special Properties of Options

Options possess unique properties that make them special trading vehicles. For starters, unlike stocks and futures, their performance is nonlinear. Every point a stock or future contract moves results in the same amount gained or lost. Their performance graph is a straight line. Figure 10.1 illustrates this point.

In contrast, an option's performance graph *curves* upward. This nonlinear shape means that as the underlying moves in favor of the option, the option makes money faster. If the underlying moves against the option, the option loses money slower. (In Figure 10.2, focus on the dotted line, which represents *today's* performance of the option.)

The real-world implication of this upward performance line curve is that an option's value can go up without limit but can only drop to zero. It is this

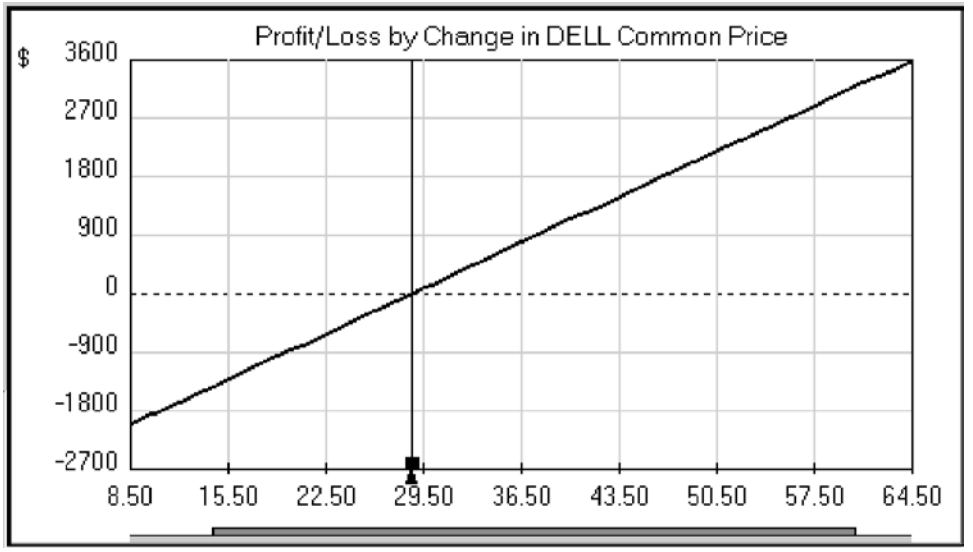

Figure 10.1 The Linear Performance of a Stock or Future

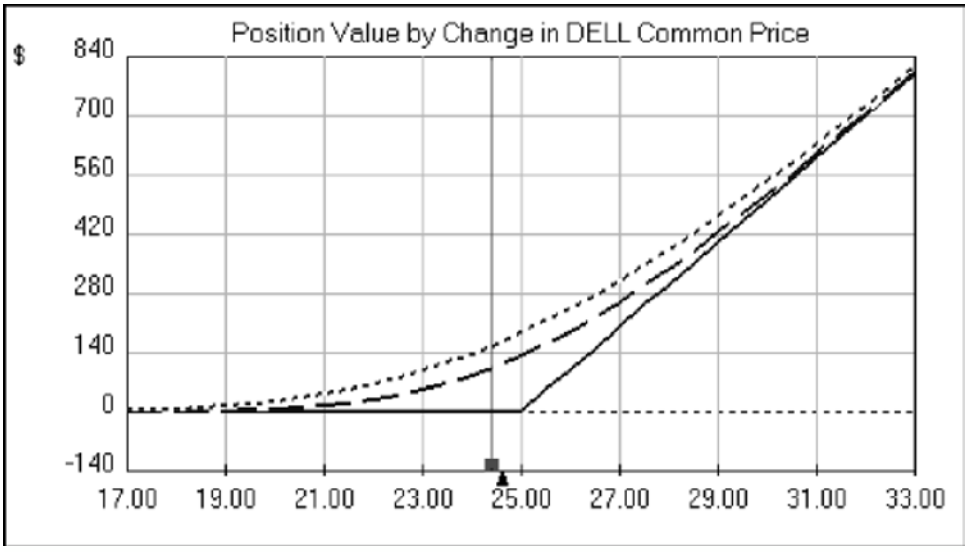

Figure 10.2 The Nonlinear Performance of an Option

unlimited profit/limited risk profile that makes options an attractive trading instrument for buyers.

The only major detractor is a little element called "time decay." As time passes, all other parameters being equal (i.e., the underlying price has not changed), an option's value falls. In our diagram, the dashed line represents the theoretical value of this call option 64 days from now (halfway to expiration), while the solid line represents the theoretical value of the option at expiration. The diagram clearly shows that if the stock fails to move above 30 (the strike price of this option), the value of this option will fall gradually and inevitably to zero.

Many options traders put this time decay property in their favor by selling options (rather than buying). Having time working in your favor can be beneficial, but it can convey a false sense of security; time is what gives the underlying a chance to move—potentially against the option seller's position.

Consequently, there is a trade-off (or risk) for both sides. The option buyer has the nonlinear performance line of an option in his favor, but time works against him. The option seller has time on his side, but the option's nonlinear performance line works against him. To limit damage due to time decay, the option buyer may choose to hold his position only for a short time. Likewise, the option seller, to limit the damage from an adverse price movement, may decide to use a stop.

Volatility Trading

The other unique property options possess is sensitivity to volatility. Options on more volatile assets (those that experience greater fluctuation in price and trading volume), all else being equal, are more expensive. Options on less volatile assets are cheaper. And the difference between the two can be significant. Even options on the same underlying asset, during a period when the asset's price is perceived by the market to be volatile, can trade at twice the price they do during quieter periods.

Volatility, then, gives options an extra dimension on which they can be traded. Not only can they be traded based on expected price moves in the underlying (called *directional trading*), but they may also be traded based on expected swings in volatility levels—buying options when volatility is low and the options are cheap, selling options when volatility is high and the options are expensive. Such trading is called *volatility-based trading*.

Options and Changing Conditions

It is essential to know how options respond to changing conditions. As we've learned, options can move in three dimensions: (1) Price, as options respond to changes in the price of their underlying; (2) Volatility, as options respond

to changes in the perceived volatility of their underlying; and (3) Time, as an option's value decays over time, all else being constant.

The initial two dimensions are tradable. Price and volatility fluctuate giving traders the opportunity to bet on their future direction. But time is different; it only marches in one direction: forward. So, while a trader may put time on his side by selling options, its basic value cannot be treated the same as the dimensions of price and volatility.

With changes in the underlying price or interest rates, the values of calls and puts move in opposite directions. For instance, when the underlying price rises, calls go up and puts go down. Volatility and time cause calls and puts to rise and fall in tandem. When the underlying becomes more volatile, both the calls and the puts go up in value. With the passing of time, both calls and puts decline in value. Table 10.4 summarizes how conditions affect option value.

The overall effect of interest rate changes on an option is very small. The effect of time decay is gradual, with some acceleration as expiration approaches.

The Greeks

Options traders use several parameters, so named the "greeks" after Greek letters, to determine how sensitive an option is to changing conditions.

The first, *delta*, measures how much an option's price moves in response to a one-point increase in the price of the underlying. For example, if an option moves up 0.5 when its underlying moves up 1.0, the option's delta is said to be 50. This means you would theoretically gain $50 per option contract.

Since calls and puts move in opposing directions when the underlying price changes, calls always have positive deltas, and puts always have negative deltas. Call options have deltas that range from 0 to 100. Put options have deltas ranging from 0 to –100. At-the-money calls typically have a delta close to 50, while at-the-money puts typically have a delta close to –50.

Table 10.4
How Different Conditions Affect the Value of an Option

	Call Options	**Put Options**
Underlying price goes up	Go up	Go down
Underlying price goes down	Go down	Go up
Volatility goes up	Go up	Go up
Volatility goes down	Go down	Go down
Time passes	Go down	Go down
Interest rates increase	Go up	Go down
Interest rates decrease	Go down	Go up

Another greek, *vega*, measures how much the price of an option changes in response to a one-point increase in volatility. For example, if an option has a vega of 29, and volatility increases from 22 percent to 23 percent, the option's price will increase by 0.29, and its value by $29. As volatility can never be less than zero, all options have positive vega.

Theta measures how much the price of an option should drop today just due to the passage of time. For example, if an option has a theta of –4, its price should fall 0.04, and its value by $4, by the end of the day. As theta deals exclusively with negative movement, all options have negative theta.

Rho measures how much the price of an option should change (+ for calls and – for puts) in response to a one-point increase in interest rates. For example, if a call option should increase 0.02 when interest rates go up one point, the option's rho is said to be 2.

The greeks are theoretical because they measure how an option *should* respond to changing conditions. The mathematical models used to calculate options' fair values also produce the greeks as by-products. That's important because as an option's fair value constantly changes in response to changing inputs, the greeks also change in a corresponding manner. In fact, there is another greek, *gamma*, that's just for measuring how fast delta changes!

Sophisticated options traders use the greeks to determine their risk, as the greeks reveal the exposure of their current position. Greeks become even more valuable the more complicated the position gets. Market makers, for instance, often hold positions (long and short) in many different options on a particular asset. By knowing the net greeks of their combined position (computed by totaling the greeks of each option position they hold, multiplied by the number of contracts they have in each option), they can determine their net risk, and make adjustments if necessary. For example, if they see that their net delta is a negative 514, they may buy 500 shares of stock to change their net delta to a negative 14—reasonably close to zero, or "delta-neutral." A delta of zero means their total position value will remain unchanged even when the price of the underlying changes.

Who Are Market Makers?

Individual investors seldom trade with each other. More often, though never knowing it, they trade with market makers. Market makers are bound by agreement with the exchanges to post bid and asked prices, and trade with interested buyers and sellers at the posted prices. By doing so, they are making a market.

An individual market maker is usually assigned to one or more underlying issues, and each underlying issue has one or more (typically more) market makers assigned to it. When multiple market makers are assigned to an underlying, they are in competition with each other. This competition can be cutthroat or friendly, like a cartel.

Market makers must trade with parties interested in buying and selling at the stated bid and asked prices, but they are not required to trade an unlimited number of contracts. The exchanges only require them to trade up to ten contracts at a time. If an investor wants to trade more than ten, he may get the stated price for the initial contracts but may have to pay a bit more (if buying) or receive a bit less (if selling) for further contracts.

While ten is the minimum requirement, the most heavily traded options markets have hundreds, even thousands, of contracts available at the posted bid and asked prices. Many quote services show the number of contracts available at the bid and the asked prices. For example, you may see a bid of 2.30, an asked of 2.50, and a number like 300×500. This indicates 300 contracts are available at the bid and 500 contracts are available at the asked.

To use a rough analogy, market makers act like bookies, taking a small piece of the action from all participants. Using a computer model to determine each option's fair value, they typically post a bid price just below fair value and an asked price just above fair value. If a seller enters the market offering options at a fair price, the market maker buys from him. If a buyer enters, the market maker sells to him. Since the market maker's selling price is higher than his buying price, he invariably earns a profit. The market, though, seldom offers the perfect convenience of a seller entering immediately after a buyer, and vice versa. Often, orders flow all in the same direction. This requires that the market maker work to hedge his position. After taking on a new long position in a call option, the market maker will immediately look for any other call options on the same underlying he can sell at reasonable prices. He may quickly sell an appropriate quantity of these to bring his delta back to near zero, or he may look for puts on the same underlying he can buy. If he is unable to find such opportunities, he will likely sell (or short) the appropriate quantity of the underlying itself.

As you can see, the market maker is always working to manage his risk. The new long call position placed the market maker in a net long (positive delta) position that exposed him to losses if the stock were to drop. Not being interested in betting on the stock's direction, he looks for a way to neutralize, or hedge, his position. Any of the earlier trades mentioned will accomplish that. Using the computer model, the market maker can accurately determine what the appropriate quantity would be.

OPTION STRATEGIES

Equipped with the knowledge of basic options terminology and some of the mechanics, the next step is to discuss options strategies. *Strategy* might sound like an overall approach to trading the market, but in options trading parlance, strategy simply means a kind of position (e.g. *short calls* or *long puts*). Options

trading strategies can be separated into two broad genres: *single-option* and *multiple-option*.

Single-Option Strategies

You may recall, there are four basic single-option strategies:

Long call
Short call
Long put
Short put

When you consider that each of these strategies could be used to complement a position in the underlying, you now have eight single-option strategies. But, before listing all eight, there are two more terms you need to know: *covered* and *naked*.

If an investor is short a call option and already owns the underlying that would need to be delivered in the event of assignment, his short call option position is covered. If he does not own the underlying, then his short call option position is naked. Likewise, if an investor is short a put option and is short in the underlying, his short put option position is covered. Otherwise, his short put option position is naked.

Thus the eight single-option strategies are:

Long call
Short covered call
Short naked call
Long call with short stock
Long put
Short covered put
Short naked put
Long put with long stock

We'll discuss each strategy, including how each performs and when you would want to use it. Along the way we'll introduce the concept of margin requirements. While our examples use stock options, these concepts are universal and apply to options on every kind of underlying asset.

Long Call

A long call is considered a *leveraged* position in comparison with owning the stock itself. The call option holder controls the stock without actually possess-

ing it. And as the option purchase is much cheaper than buying the stock, the option holder can control the stock for a fraction of the cost. If the stock's price goes up even just a few percentage points, it's likely that all its call options will increase by a greater percentage; some may even double in value.

Other important characteristics of the long call are *limited risk* and *unlimited potential*. An option buyer can only lose the amount paid for the option, nothing more. At the same time, upside potential is theoretically unlimited. A call option's value will continue to increase as the price of the underlying continues to go up.

Leverage, limited risk, and unlimited potential make call option buying attractive to speculators looking for quick upside advances. I say quick because, remember, the option buyer's enemy is time decay. For each day the call option position is held, it loses some of its value. That's why long calls are best for speculators expecting an upside move within a few days at the most.

The profit diagram for the Long Call (Figure 10.3) illustrates the strategy's limited risk/unlimited potential profile. The three lines illustrate how time decay affects the position. The dotted line represents the long call's theoretical performance as of today, while the solid line represents the performance at expiration (the final day of trading), and the dashed line represents the performance at the midway point between today and expiration.

Short Covered Call

We discussed this strategy, also known as a *covered write*, or *buy write*, earlier with the car example. It requires the covered call writer to give up control over

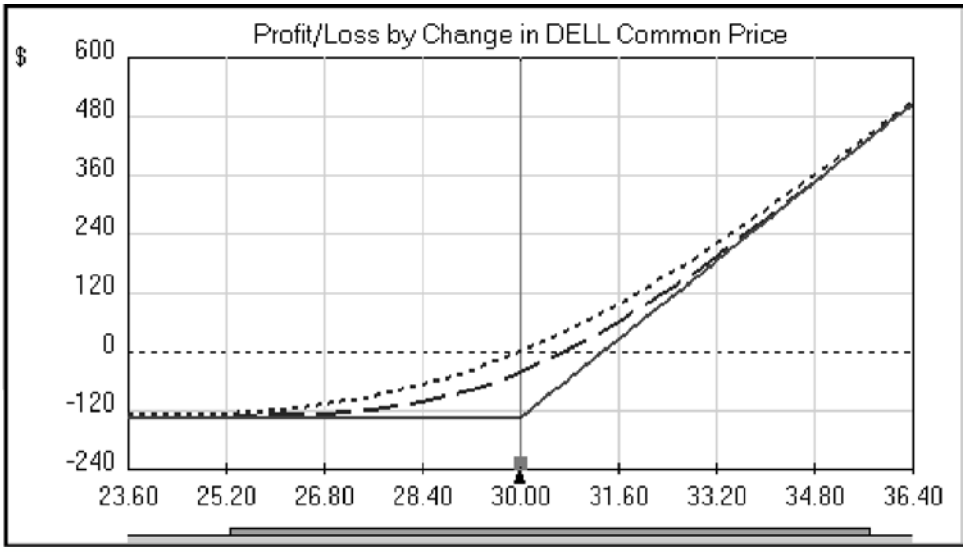

Figure 10.3 Profit Diagram of a Long Call

his stock for a limited time in return for income from the sale of the option. This income is his to keep regardless of what happens.

While the speculative call buyer will only hold his position for a short period of time, the covered call writer often expects to hold his position to expiration. If the option is out-of-the-money at that time, it expires worthless. If the option is in-the-money at expiration, the investor can do two things: He can buy the option back to close the position and keep his stock; or, he can do nothing and see his asset *called away* (loses his stock). Some investors keep their stock and repeatedly sell covered calls after each expiration, thus continually adding to their income. The advantage is that covered writers make money during periods when their stock holdings are going nowhere. The income that can be generated can be impressive—upward of 40 percent annual returns is typical. Remember, though, such returns are possible only if the stock goes up or remains where it is.

Covered writing is a conservative strategy. The sale of call options against stock holdings reduces the overall variance of returns, thus reducing risk in the traditional sense. This is why options were created in the first place (not for speculation). Still, speculators are an important part of the market. They assume risk that portfolio managers would like to offload. Thus, options are an essential mechanism for risk transfer—from those who don't want it (hedgers) to those who do (speculators).

While covered writing is attractive because of the immediate income it generates, it does have its shortcomings. The covered writer has the same downside risk, as with just owning the stock, and his upside potential is limited. The Short Covered Call profit diagram (Figure 10.4) illustrates the downside risk and limited potential.

Short Naked Call

The naked call seller has the same interest as the covered call seller—income. The difference is that the naked option seller does not have a position in the underlying. If assigned, he must buy shares on the open market to deliver them. The risk, of course, is that he'll have to pay the market's then-current price.

Performance is the primary difference between being short a naked option and being short a covered option. Without stock to complement a naked short call position, there is nothing to offset the position's risk if the stock goes up.

By selling a naked call, you get precisely the opposite performance characteristics from buying a call: *unlimited risk* and *limited potential*. An option seller can receive the amount he was initially paid for the option, but no more. At the same time, he has theoretically unlimited risk. As the price of the underlying climbs (with no limit), the call option's value climbs. At some point, you need to buy that option to close the position (or have to acquire stock to deliver when assigned, resulting in almost an identical monetary result).

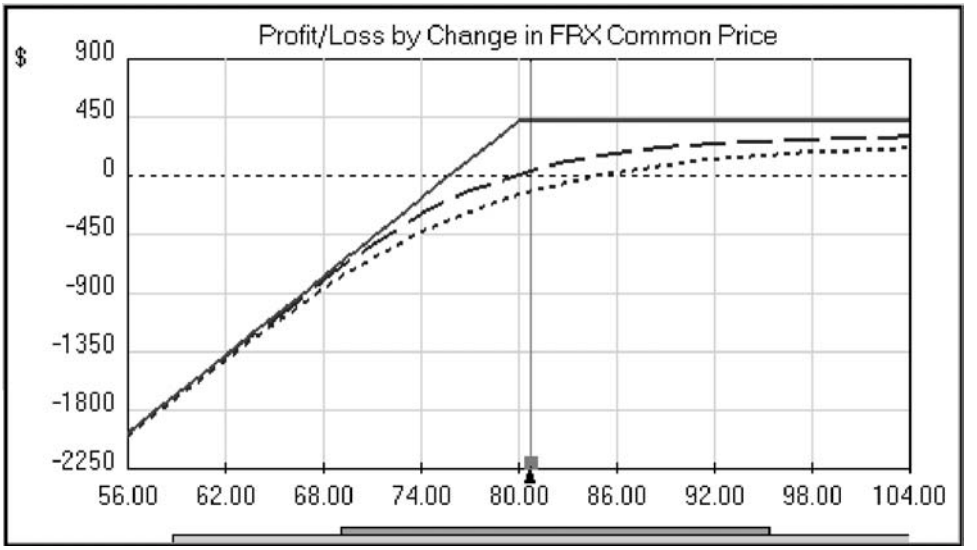

Figure 10.4 Profit Diagram of a Short Covered Call

Still, investors like the prospects of earning time decay dollars (in a bearish to neutral position) when the position costs less to put on than a covered write. In fact, after getting the credit following the initial sale of the option(s), why wouldn't a trader continue putting on large positions—as large as they want? Well, they can't, because brokerages require investors to keep money in their account to cover potential losses. This coverage is called the *margin requirement*. (Margin requirement is actually old terminology. The new term, *performance requirement*, is closer to the mark, but the old term still lives on.)

There is a standard formula for computing the requirement for a naked short option position. Without going into details, the amount required is roughly 5 to 20 times the credit received from the sale of the option(s). So, if you sell an option and receive a $1,000 credit, you will be required to put up anywhere from $5,000 to $20,000 in collateral to support the position.

Naked option writers rely on stops to help control the theoretically unlimited risk. That works well, provided the underlying trades continuously (i.e., its price doesn't jump dramatically). As many investors have seen, however, a stock can sometimes close at one price and open the following day at a radically different price. Stops are useless against this type of risk. To limit this exposure, some traders stick to writing index options, since index options don't possess nearly this level of risk. The profit diagram for the Short Naked Call (Figure 10.5) illustrates this strategy's unlimited risk, and limited potential. As you can see, it makes money if the underlying price stays the same or goes down.

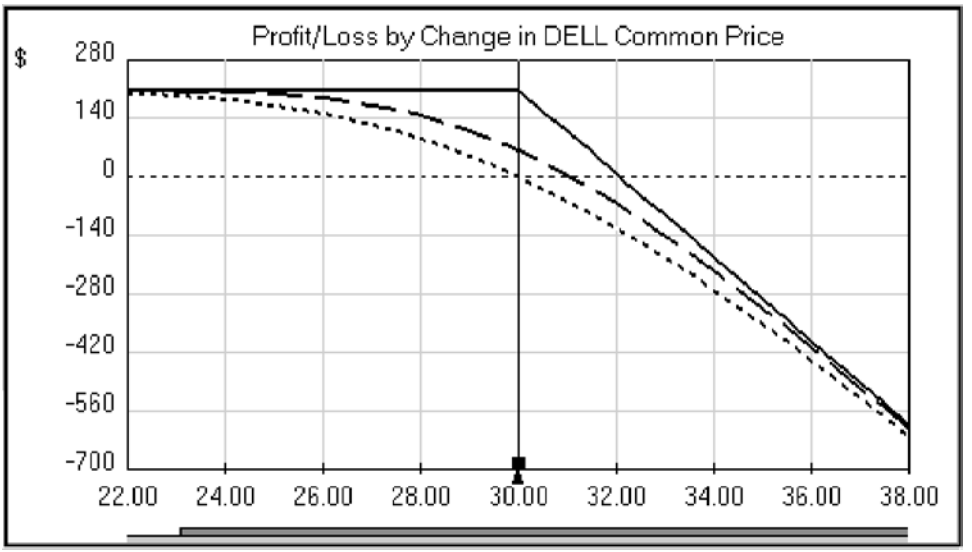

Figure 10.5 Profit Diagram of a Short Naked Call

Long Call with Short Stock

This is a rare strategy that is seldom used or even discussed. To protect a short position in stock from an adverse move (to the upside), the trader buys calls. This permits participation in the expected downward move, and alleviates concerns about the stock jumping northward. However, it costs money to buy those calls, and that subtracts from the position's overall profit. And after the calls eventually expire, if the investor wants continued protection, he'll need to buy more. The profit diagram for the Long Call with Short Stock (Figure 10.6) illustrates this strategy's limited risk and unlimited potential.

Long Put

The put option holder maintains the right to sell stock without actually possessing it, and does so at a fraction of the cost of shorting the underlying itself. Buying a put is a highly *leveraged* bearish position. When the stock price drops even by a few percentage points, its put options will likely increase by a greater percentage; some may even double in value.

The Long Put has the same characteristics as the Long Call: *limited risk* and *unlimited potential*. An option buyer's risk is limited to the amount paid for the option, and no more. On the plus side, the profit potential is theoretically unlimited. A put option's value rises without limit as the price of the underlying goes down. Well, it's not absolutely unlimited potential, as the farthest the pot could increase in value is if the stock goes to zero.

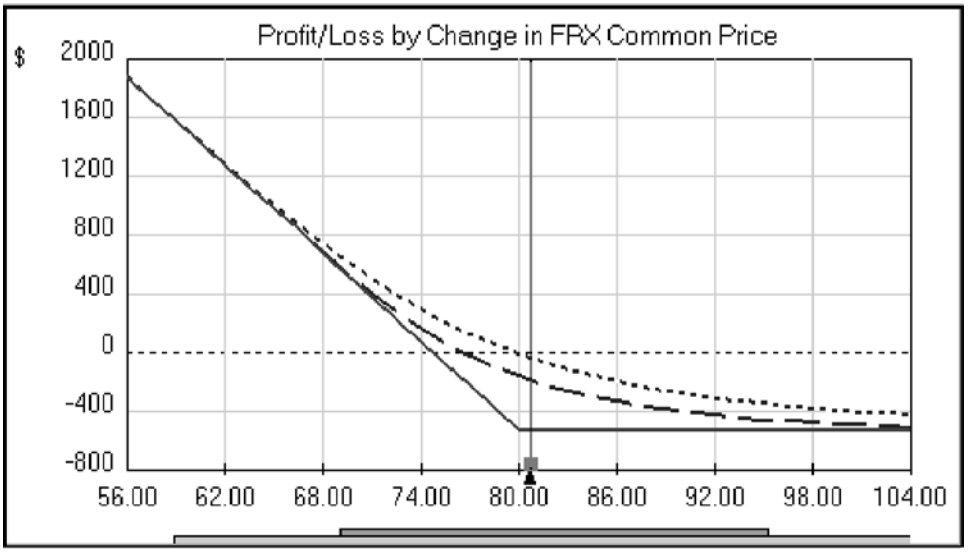

Figure 10.6 Profit Diagram of a Long Call with Short Stock

As with buying calls, the qualities of leverage, limited risk, and unlimited potential make put option buying attractive for speculators betting on a quick downward move in the underlying. Again, the buyer's enemy remains time decay, so put buying is best suited when the speculator believes a downward move will occur fairly quickly. The profit diagram for the Long Put (Figure 10.7) illustrates this strategy's limited risk and unlimited potential.

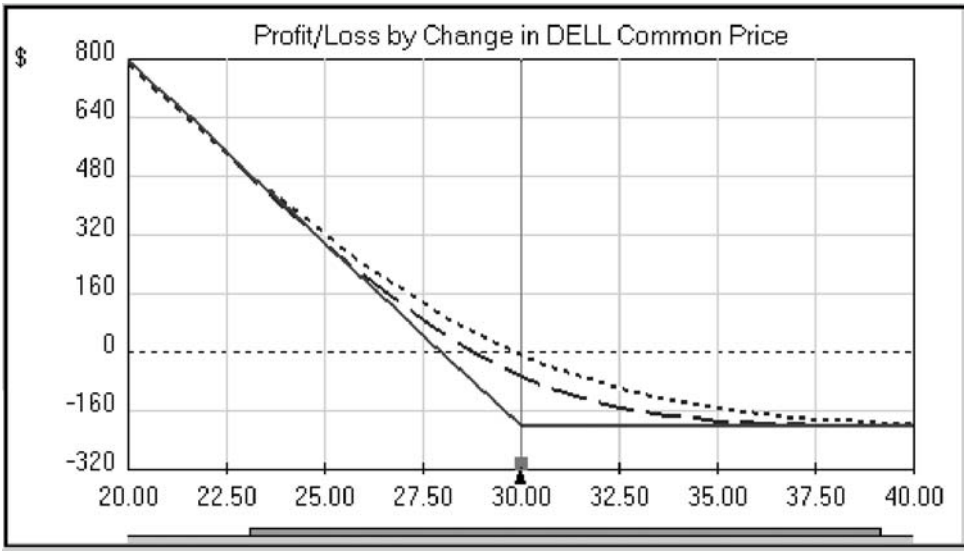

Figure 10.7 Profit Diagram of a Long Put

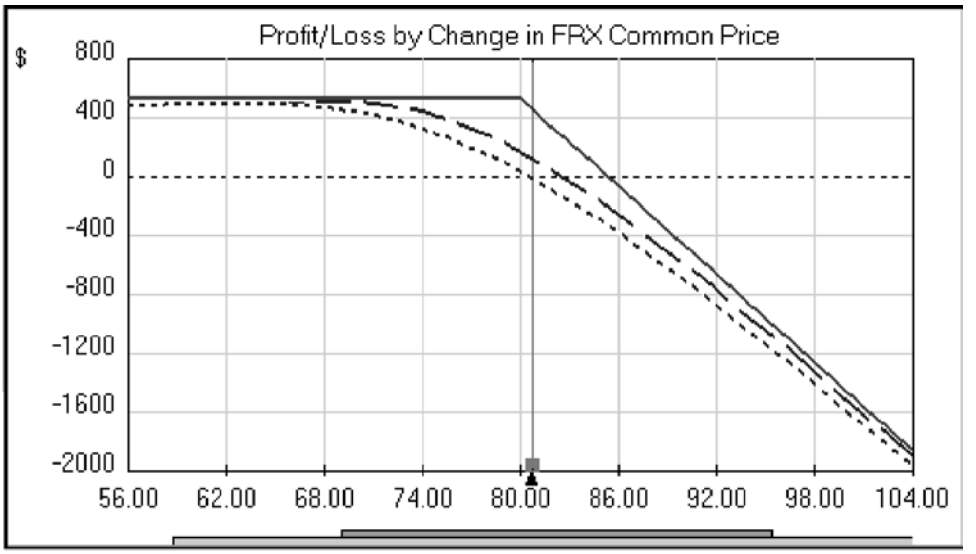

Figure 10.8 Profit Diagram of a Short Covered Put

Short Covered Put

This strategy is so seldom used or discussed that the term *covered write* is universally understood to mean buying stock and selling calls. But from a technical perspective, shorting stock and selling put(s) is also considered a covered write. The short stock covered writer releases control over his short stock position for a limited time in return for income resulting from the option sale.

The characteristics of this strategy are an identical mirror image (in terms of stock price direction) of the traditional covered write (Figure 10.8).

Short Naked Put

The naked put option seller has one interest: income. With this strategy, the investor expects the underlying stock to remain neutral or go up, thus allowing his short option to expire worthless. If the stock drops, he's likely to be assigned. If assigned, he'll instantly be long the underlying stock. The price he'll pay for the assigned shares, of course, is the strike price of the puts. So, this investor often shorts out-of-the-money puts at a strike price where he would consider the stock purchase a bargain.

Selling a naked put offers precisely the opposite performance characteristics as buying a put: *unlimited risk* and *limited potential*. It's not *absolutely* unlimited risk, as the farthest your short put can be driven into the money is if the stock goes to zero.

Even with the risk, investors are attracted to the prospects of earning time decay dollars in a bullish to neutral position. As with selling naked calls, your account is credited for the initial sale of the option(s) and your brokerage requires collateral to cover the position.

If they are not interested in owning shares of the underlying, naked put writers employ stops to help control the risk from a downward move in the stock. That works as long as the underlying trades continuously. But again, a stock can sometimes close at one price and open the next at a very different price, and this kind of risk cannot be controlled using stops. The Short Naked Put profit diagram (Figure 10.9) illustrates this strategy's unlimited risk, limited potential, and reveals that it makes money if the underlying price remains the same or goes up.

Long Put with Long Stock

This strategy, sometimes called *a married put* or *protective put*, is considered the most effective way of hedging long stock positions, as it absolutely limits downside risk. Fund managers, unwilling to sell their stock (often for tax reasons), will buy puts to see their underlying positions through what they believe will be rough periods. It costs money to buy those puts, and that protection subtracts from the fund's overall returns. And after the puts eventually expire, if the fund manager wants continued protection, he'll need to invest in more puts. The profit diagram for Long Put with Long Stock (Figure 10.10) illustrates this strategy's limited risk and unlimited potential.

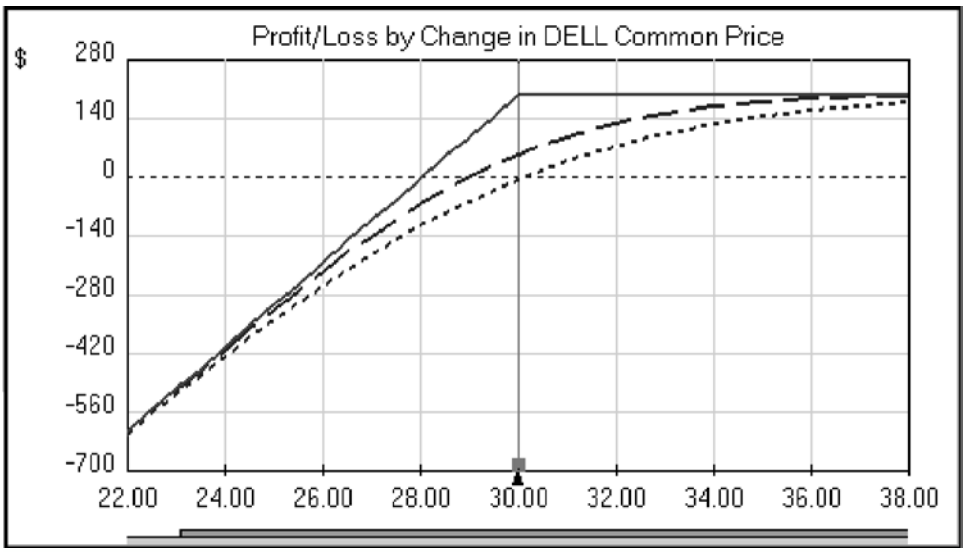

Figure 10.9 Profit Diagram of a Short Naked Put

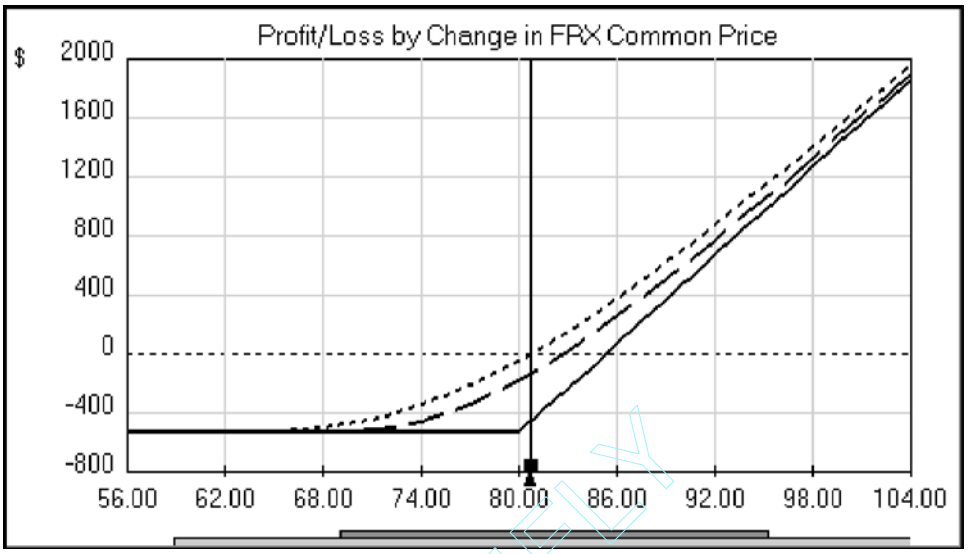

Figure 10.10 Profit Diagram of a Long Put with Long Stock

Equivalent Strategies

If you think you've been seeing some of the same profit diagrams repeatedly, in fact, you have. The performance curve of an option is one basic shape—flat to the out-of-the-money side sloping 45 degrees to the in-the-money-side.

The performance curve of a Long Put is the same as that of a Long Call, only mirror-imaged left to right. The performance curve of a short option position is the same as that of a long option position, only mirror-imaged top to bottom.

The performance curve of a single position in the underlying is a straight line at 45-degrees. When a complementary stock position is added to an option position, the stock's 45-degree line combines with the option's line to "rock" the line so the flat line portion becomes a 45-degree portion and the 45-degree portion becomes a flat line. Consequently, there is one basic shape, and only four ways of showing it—flipped top to bottom and/or flipped left to right.

Therefore, each of the eight single-option strategies has a counterpart with an identical performance curve. For example, Long Call and Long Put Long Stock have identical curves. Table 10.5 summarizes the equivalent strategies.

How does one decide whether to use a single-option strategy or its counterpart involving the stock? If one currently holds a stock position he wants to keep, the best approach would be to add the appropriate option to the position. But what should be done if the investor is constructing a new position from scratch?

Table 10.5
Equivalent Strategies

Long Call	Is equivalent to	Long Put Long Stock
Short Covered Call		Short Naked Put
Short Naked Call		Short Covered Put
Long Call Short Stock		Long Put

A position involving the stock will likely require more capital, and involves more transactions—two to get in and two to get out. Also, shorting stock has its limitations. Therefore, the single-option strategy is often preferable because it usually accomplishes the same objective with fewer resources.

Combinational Strategies

The previous single-option strategies section was covered in a formal manner for the purpose of planting the shape of each single-option performance curve firmly in your mind. While the more complex option strategies properly belong to a more advanced options book, I would be remiss if I didn't provide a glimpse of the possibilities offered by combinational option strategies. When we put two or more options together, their combined performance curves form new and interesting shapes. Common strategies involving two options, with names like *vertical debit spreads*, *straddles*, *strangles*, and so on, also allow the options trader tremendous flexibility in meeting her investment objectives under various market conditions.

Contrary to what you may hear or read from time to time, there is no single strategy that automatically makes money. New options traders often enter the market with the belief that there exists one magical combination that produces positive returns over the long term. What they fail to understand is that since each (fairly valued) option is a net zero expected return item, no matter how many of them you put together, you still have a net zero expected return (in the absence of a directional or volatility change exception).

Since no particular strategy is suitable for every opportunity, the trader needs to be able to apply the most appropriate strategy in the given situation. This requires familiarity with the various strategies and their performance characteristics. Tables and diagrams can be constructed to show which option strategies are bullish, bearish, aggressive, moderate, or neutral. Such tables and diagrams are useful, but sometimes fail to account for three primary considerations: price direction, time frame, and volatility. Another difficulty is that many strategies overlap others in how and when they should be applied.

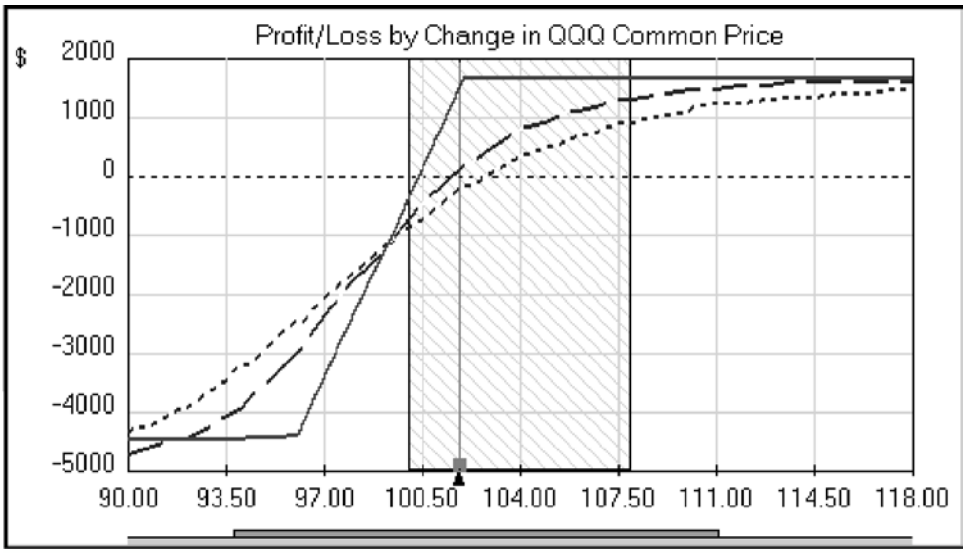

Figure 10.11 Profit Diagram of Put Credit Spread

So, understanding how the various strategies perform is best gained through experience, and may be accelerated by the use of models and software that can simulate the performance of any strategy in the three dimensions (price direction, time frame, and volatility).

Combinational strategies involve taking positions in two or more different options, and possibly another in the underlying. Like combining hydrogen and oxygen to form a unique substance (water), putting options together in various combinations results in some amazingly unique risk/reward profiles.

For example, selling a naked at-the-money option is a very risky strategy. And purchasing an out-of-the-money option is costly and has a poor probability of success. However, combine these trades (using two options of the same type and in the same expiration month) and you form a credit spread—one of the safest and most successful option strategies around. (See Figure 10.11 for an example of a put credit spread's profit/loss profile.)

Likewise, when you buy an at-the-money call or put, chances are you will lose all your money (stops notwithstanding). However, buy a straddle (both a call and a put at the same strike price and expiration month) and the possibility of losing all your money is practically nil (the underlying would have to finish precisely on the strike price). (See Figure 10.12.)

Many people who know about, but are not truly familiar, with options believe they are inherently risky. While options allow you, and even tempt you, to take speculative chances, it is not true that they are inherently risky. Options are enormously flexible. Yes, they allow you to speculate, but they

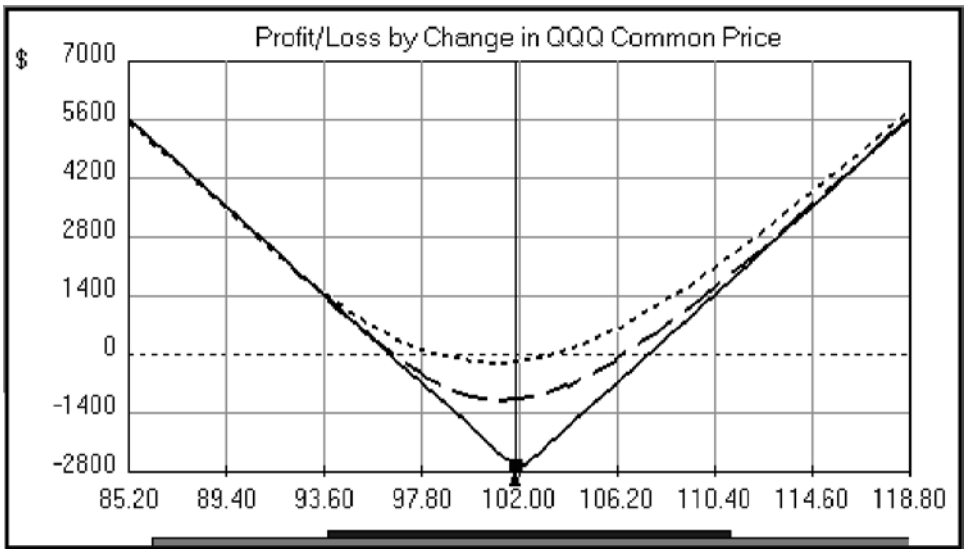

Figure 10.12 Profit Diagram of a Long Straddle

also allow you to hedge or even simulate a portfolio. Through combinational strategies, a position can be constructed that closely fits your goals, price predictions, projected holding period for the trade, and the current volatility environment.

11

Interviews with Developers

Over the years, Futures Truth Company has interviewed some of the best minds in the system developing and trading industry. We thought it would be appropriate to conclude with their interviews. We believe you will find wisdom and some good ideas to incorporate into your own trading ideas and systems.

Welles Wilder

Age:	63
Education:	B.S., Mechanical Engineering
Current Position:	Director of the Delta Society
Favorite Book:	The Bible

Published in late 1998

As the first, we thought it appropriate to choose someone whose ideas originated before computers were widely used by individual traders but who had a lasting affect on both technical analysis and, later, computerized trading.

His tools were easily used in paper spreadsheets and modern computerized trading programs, during the past two decades since something called the programmable calculator gained acceptance. Much of this person's studies culminated in the writing of his technical analysis book, *New Concepts in Technical Trading Systems*, published in 1978, in which he described a tool he created called The Relative Strength Index.

The Relative Strength Index (RSI) became one of the most widely used trading tools. The renowned technician we are describing, Welles Wilder, spoke to Steve Toney recently from his home in New Zealand.

■ ■ ■

How did you get into the futures business?

After a ten year career in mechanical engineering, real estate, and land development, I sold my interest in over a thousand apartments and various other real estate projects and began to pursue other areas of interest.

I read a book titled *Silver Profits in the Seventies*, by Jerome Smith. Since real estate is a highly leveraged situation, I looked for a way to buy silver in a highly leveraged situation; this led me to the commodity futures markets.

I made a lot of money in silver but lost most of it in learning to trade other commodities. This led to about five years of reading, and researching everything I could get my hands on relating to futures trading using mathematical models. In 1978, I published the results of these studies in *New Concepts in Technical Trading Systems*. My life has not been the same since. (*Note: Welles wasn't too far off from his prediction. He may have missed Y2K, but he was right on the money with the markets after 1999.*)

What do you think about Y2K and the chances of disaster?

At first I thought it was just a bump in the road causing a lot of hype. However, because it presented such enormous possibilities, I began to study it in depth. (Some of my friends call me "Bulldog" because when I latch onto a concept, I don't let it go until I have exhausted virtually all the potential it contains.)

In mid 1996, not much had been published on the subject (of Y2K), so the main area for research was the Internet, which came along just in time. I always enjoy research, because it involves sifting through opinions and latching onto the parts that can be proved or, in many cases, the items that cannot be disproved. The more I studied, the more non-disprovable information I gathered, and the more concerned I became.

I believe most of your readers are most interested in the financial repercussions of Y2K, so I will concentrate on that arena.

The greatest financial party of the century is almost over. Banks and investors have become drunk with their huge profits and have devised more and ingenious ways to monetarize debt and then leverage it by means of derivatives.

The fractional reserve banking system now has, on average, $1.32 to dispense to depositors for each $100 in deposits. The stock market is now more "out of value" than it was in 1929. Bankers see no place for gold in the monetary system; no one remembers or even considers deflation.

Y2K will be the catalyst that brings this house of cards crashing down. It will happen this year in 1999. It started in Asia, and the dominoes are falling. Russia has defaulted. South America, Japan, Mexico, China are all having big problems. Savvy investors are using this last market up-move to get out.

I believe that by mid summer to late summer the perception of the implications of Y2K will start bank runs throughout the world and in the United States. If the U.S. stock market has not already started to plummet, this will bring it about. I believe it will make 1929 look like a walk in the park. I believe 1999 will be the beginning of sorrows.

Where are the best financial opportunities now?

In light of the above, there are basically two avenues. One is to buy put options on the stock market indexes. The other is to buy gold and silver call options. I prefer the latter because they are cheaper, and I believe they will provide more return on investment. Also, when the bank runs start and the government imposes restrictions on withdrawals, what will the average guy write a check (on) in order to (acquire goods)—it will be on the only things that have always had intrinsic value, gold and silver.

It would only take a small fraction of these "average guys" to run the price of gold and silver to unheard of levels. Are the bankers still going to think of gold and silver as a barbaric relic and continue to sell their gold? No. They will suddenly realize they have pushed the envelope too far and that the confidence in the reserve system has finally been broken. It will be a new ball game with new rules: "He who has the gold makes the rules." The bankers will be first in line at the gold window to try to protect their own wealth. I believe this will happen in the late summer or fall of 1999.

I recommend buying the December 1999 390 gold call options; currently they are under $200 apiece. If gold goes as high as it has been in the past, each call option will be worth $40,000. Regarding silver, I recommend buying the December 1999 silver call options at a strike price of $8. These are currently available for around $400 each (call prices as of late January).

Are you still actively trading?

Yes, but from a hands-off vantage point. I have a person in my office who is instructed to trade my account in strict accordance with the Delta Member's System. Neither he nor I are permitted to deviate from the system. This is the best system that I know.

Who do you think has good ideas now, specifically regarding technical analysis?

I must admit that in recent years I have not kept up with all the new stuff that has come down the pike. I think Tom DeMark leads the pack in coming up

with original ideas. If you will allow me to include myself in this category, I have spent the last two years in researching a way to automatically denote a trend change.

Explain what the Delta Society does?

In 1983, I founded the Delta Society International as a vehicle to distribute information on futures market turning points to its members. A man named Jim Sloman made an amazing discovery. He showed me that there is a perfect order in all freely traded markets. Even though the order is perfect, the accuracy of the projected turning points is not perfect—if it were I would be wealthier than Bill Gates, and no one would have ever heard of the Delta Society—however I have researched the historical occurrence of each Delta Turning Point, and have defined the standard deviation and the 100 percent range for each point.

I have now been able to provide Delta Members with a mechanical way of utilizing this information. I consider that to be my best work and final achievement in the arena of technical analysis.

Are you ever surprised at the following your ideas have created?

Frankly, I haven't thought a lot about it, but the answer would be "Yes." However, when I look back over the last 25 years, I must admit that I could never have chosen any career that has been as challenging and interesting, as rewarding, and half as much fun as my third and last (this) career. I feel very fortunate indeed.

Dr. John Clayburg	
Age:	52
Education:	Ph.D., Veterinary Medicine, Iowa State University, 1971
Positions Held:	Independent System Developer, Custom TradeStation Programmer
Favorite Books:	Anything by Tom Clancy

Published in 1999.

Two systems developed by the same person have popped up on the top ten systems list regularly in recent issues of Futures Truth: Feeder Trader and Cyclone, which is an S&P day trading system.

While S&P futures traders are obvious target customers for a systems developer, Feeder Cattle is a thinly traded market that does not attract a lot of interest in the trading world; Feeder Trader's strong hypothetical performance might change that though.

For John Clayburg, because he is a veterinarian and a trader, foreseeing patterns in both the feeder trader market and the S&P came naturally. Clayburg created Cyclone and Feeder Trader.

Futures Truth spoke to Clayburg recently, as part of a continuing series of interviews with those who have greatly impacted futures systems trading.

■ ■ ■

What do you think are the hot markets this year?

I seem to concentrate on only two-to-three markets now and really don't have a valid opinion here.

What inspired you to create the Feeder Trader and Belly Trader systems?

I routinely scan all markets for the best responses to certain repetitive patterns. The feeder cattle market comes up regularly in these scans, since it seems to be more cyclical than some others. Also, since we raise feeder calves on our family farm, the feeder cattle market was a natural for us.

As for Cyclone, stock index futures seem to create the most interest. It is probably the most challenging market to handle—and I rarely turn down a challenge.

Do you plan any revisions to your systems anytime soon?

I have made only one change to Cyclone since its release, in response to the increase in volatility in the S&P index in 1997. I have made no changes to the Feeder Trader Program. I plan no changes in either program.

You must be a very busy guy. Does futures trading and programming ever interfere with your life?

Only if I let it, which is easy to do. There's always one more project—one more good idea. . . .

Are you worried about the generally poor performance of S&P daytrade systems during the first quarter of this year?

Unhappy, yes. Worried, no. Cyclone was developed, as was probably most other systems, by the careful observation of repetitive patterns in the S&P market. The most rewarding, repetitive pattern in this market is the big trading day in which the trend develops early and persists for most, if not all, of the day. During the time to which you refer, the market was markedly devoid of the usual presence of these typical days. This resulted in frequent stop-outs as the system tried, but failed, to catch these big trending days.

Obviously, a system designed around a particular pattern will not work well when this pattern fails to repeat itself as in the past. Why this change in market personality occurred is the big question. It's possible the Dow's flirtation with the 10,000 psychological level was to blame. You can look back over the historical price patterns of this market and note several periods where the big day vanished. The period we have just experienced has been about the length of those noted on past charts. The market seems to be returning to normal as we do this interview.

If history repeats itself, these systems will return to their past performance as the market regains its normal personality one again.

What advice do you have to those who trade your systems or any other system?

You hit my favorite subject here. Anyone who is contemplating trading a mechanical system owes it to himself and the developer to become intimately familiar with the system before trading it. It is easy to look at a draw down figure and say, "that's no problem"—but when the draw downs hit, and they will, it's quite another thing to be able to deal with the losses. It's human nature to dwell on the profit possibilities of a system and down-play the potential for loss.

Also, those who are unsuccessful with a system are usually those who think they can beat the system by altering its rules or parameters or selectively deciding which trades to take and which not to take when they are generated by the system: If you are into system development, by all means concentrate your efforts toward building your own successful trading strategy. On the other hand, if you want to trade a particular system, trade the system exactly as it was meant to be traded by the developer. Rarely, if ever, do the two combine successfully.

What else do you do for fun, besides programming?

As our five kids begin to scatter about at the universities and eventually into their careers, we find ourselves traveling to visit the gang and getting to see more of the country in the process. Also, we enjoy raising cattle here at our home and observing the multitude of wildlife so abundant in our area.

Keith Fitschen

Age:	52
Education:	M.S., Electrical Engineering (Stochastic Estimation)
Positions Held:	System Developer for TradeSystem, Inc.
Favorite Book on Trading:	*Cybernetic Trading Strategies*, by Murray Ruggiero

Published June 1999

For this issue's interview, we talked to a systems creator who has throughout his life seen opportunity where most others have chosen to see only danger. Aggressively pursuing opportunities, but with a cool head, Keith Fitschen developed and now trades, Aberration, a computerized futures trading system that has had an excellent performance record since its release date. Keith talks about his experiences trading Aberration and his bold embracing of foreign futures markets.

■ ■ ■

What do you think the hot markets are this year?

I'm a trend-follower, not a predictor. So far, we've had good trades in the currencies (D-Mark, Swiss Franc, French Franc, and Dollar Index), all the energy products, and some good short trades in the agriculturals (wheat, beans, bean oil). Though cocoa isn't a good market for trend-following, Aberration is ahead over $8,000 in the current short trade. The industrial metals also had a good upside run (London Copper, Aluminum, Aluminum Alloy, and Nickel). The agriculturals are set up for a huge move to the upside should weather problems develop.

How did you get into systems development?

I was in the Air Force splitting assignments between engineering and flying when I realized I needed to look to the time when I'd retire and do something else. I knew that the markets, both stock and commodity, were noisy time-series data and I knew from my engineering background how to characterize and analyze that type of data. I started with commodities because they were more highly leverageable and because everyone seemed to be afraid of them—opportunity. I bought some commodity data and started to work. It wasn't nearly as simple as I thought it would be. And I'm still learning all the time. But for those with the passion, it's very rewarding.

What inspired you to create Aberration?

I started with complex analysis techniques, figuring if it was easy, everyone would have found something. I quickly learned that complex wasn't better. When I went back to the simple things analysts use, to look at data, I "found" Aberration. That was in 1986. Since then I've developed four other really good trend-following methodologies. All are relatively simple.

What do you say to those that say Aberration has had—though strong performance—a high draw down?

I hear that all the time, and it is based on the Futures Truth numbers for the seven commodities you report on. First, your rating metric (return on three times margin) is risk adverse. It is purely a profit metric. So the commodities I chose in 1993 were chosen to maximize your metric, not draw down. By comparison, our portfolios are diversified and designed for a good return and low draw down. People that see and trade our portfolios are very happy with the draw down characteristics. Second, trend-following systems give back open equity profit at the end of a trade, most systems as much as 50 percent. This open equity giveback is draw down when computed from an equity curve.

The example I always use is 1994 coffee. We entered that trade at about 88 and it went as high as about 260. Aberration got out at about 200. The 260 to 200 is a 60 point draw down, or about $23,000. But a trader wouldn't have experienced that draw down unless he was in the trade. And if in the trade, he would have made about 112 points of profit, or about $40,000. True he would have seen the draw down, but it isn't as though he experienced it when he first started trading the system. That's the problem.

People look at draw down as the amount, plus margin, that they need to trade a system. And for trend-following systems, it isn't anywhere near that amount. That's why our software reports "start-trade" draw down. That's the maximum amount a trader would go below starting equity in the history of that commodity. For small traders, that's the metric they really need to see to determine the risk with various account sizes.

Have you been trading this year?

Yes, I couldn't in good conscience sell a system unless I traded it. Last year was a good one for our portfolios, especially the smaller ones that most buyers trade. And this year is well ahead of pace for most of the portfolios. But every year will see equity rallies and draw downs with Aberration, or any other system. The key to trading is to build a plan around the performance of your system and recognize when performance, both good and bad, is within the bounds of historical. When it dramatically steps outside those bounds, the cause needs to be determined and steps taken.

What is unique, if anything, about any foreign markets you trade?

I'm a big believer in diversification, so I'll trade any market that is unique (lowly correlated with other markets). I trade things most people have never heard of: dried cocoons, the Baltic Freight Index, palm oil, azuki beans, and raw silk. Though we have great markets in the United States, there are great markets in other countries as well. I trade a lot of world bonds (Canadian, French, Italian, Swiss, Belgian, Spanish, German, Japanese, Australian, New Zealand) and I'm always looking for overseas markets that trade better than ours. For example, London and Paris sugar trade better than our world sugar, for most trend-following methodologies. And one of our weakest trading groups for trend-following is the metals. It turns out that the London Metals Exchange trades a number of metals that trade very well with trend-following systems.

Are you currently working on any programming projects?

Yes, we will be releasing an S&P system called ASCEND-X in the next month. The system is really two systems for the price of one: a daily-bar system with trades that average about six days, and an hourly-bar system with trades averaging three days. One system is countertrend and the other trend-following, so their equity curves are lowly correlated. The unique thing about them is that they make enough profit-per-trade to trade an E-Mini S&P with.

Randy Stuckey

Age:	56
Education:	B.S., Quantitative Business Analysis
Positions Held:	System Researcher/Developer
Favorite Book on Trading:	*New Concepts in Technical Trading Systems,* by Welles Wilder

Published in August 1999

Understanding how markets work and why individual market players make certain decisions is a tall order. One could argue that an education in process modeling and statistics though, along with some appreciation of the complexity of human nature, would be a darn good start.

Quality engineers research and test models that try to account for how complex human endeavors—like manufacturing automobiles, for instance—work and what variables affect the desired outcome of the process.

Most developers have come from myriad of other occupations. Probably no trading systems developer has a more appropriate career background than Randy Stuckey, a manufacturing quality engineer. Making manufacturing models helped Stuckey understand how the complex human endeavor of trading futures works.

We spoke to him recently about his systems, and probed his brain for pearls of wisdom, as part of our continuing series of interviews with important systems developers.

■ ■ ■

What do you think the hot markets are this year? Do you have any inflation, or economic disaster forecasts?

To my knowledge, no one can predict the future. As in the past, the markets that end up trending will be the big winners for the year. So far this year, all of our systems have done well for most of the currencies. Catscan continues to do pretty well with the T Bonds. Golden SX took some nice crude oil profits and Millennium 2000 currently has over $89,000 in its open positions, so this looks like another good year.

While we can't predict the future, there are several markets poised for some explosive moves. The stock market reminds me of the famous Holland tulip fiasco. Eventually valuations are going to have to get back in line with earnings reality. Either earnings are going to have to double or stocks prices are going to have to drop. Our proprietary S&P system (not for sale at this

time) seems to be preparing for a long-term short position. Inflation will probably increase late this year, which should finally allow our Millennium 2000 system to take profits on its short gold position.

How did you get into systems development?

I got involved for similar reasons to those that caused the formation of *Futures Truth*. A friend had purchased ten trading systems. None of them worked. He knew I had a background in Statistics and asked if I could determine why they were not profitable. Of course at the time I didn't have a clue, but the idea of mechanical trading systems was so intriguing that I continued to learn, eventually becoming a full-time systems researcher/developer.

By the way, now that I know why those ten systems didn't work, it's enlightening to look at a breakdown. Four of them didn't work because the system vendor literally falsified the system's performance statistics. That's felony fraud, but nevertheless that was the reality. Interestingly, Futures Truth was tracking two of the falsified systems at the time. While the vendors falsified statistics that said the systems were making 125 percent per year, Futures Truth was showing them losing 30 percent to 40 percent per year. If my friend had been aware of Futures Truth, he would have saved a lot of money. The remaining six systems didn't work for two reasons. Two of them failed because they were just plain lousy systems. The remaining four didn't work because they were grossly over-optimized.

What inspired you to create Millennium 2000, your latest system?

Some Catscan and Golden SX owners had increased their accounts quite a bit, which would justify trading more commodities. At the same time, I was looking at the effect of trading not just a diversified basket of commodities but also trading more than one system (assuming the system did not use the same trading principle). The effect of multiple system/multiple commodity portfolios was very exciting. The effect on draw down, profits and Sharpe ratio was excellent. With these thoughts in mind, I decided to develop another system to provide more diversification. The result was Millennium 2000, my first pure reversal system. It uses exceptionally simple rules, which I like.

Do you encounter many people trading, say Catscan, who are tempted to take profits early or decline trades that the system signals?

It happens all the time. Someone said fear and greed drive the commodities markets. I think this is partially true. I suspect fear is the stronger of the two. It sure seems that way as I've observed people consistently taking early profits. We've seen people take $5,000 early profits on Catscan trades that eventually were closed out with over $20,000 in profits.

What advice do you have for them?

I remind them that they are not trading the system if they take early profits or decline certain trades. Even more important they are, at that point, trading a completely untested system with hazy, unwritten rules. I further suggest that they may be better off having a broker trade the system for them.

Is fundamental analysis ever fruitful?

I suppose it has its place, though I've never met anyone who was successfully trading strictly from fundamentals. I used to create models of manufacturing processes. We then used those models to improve and control those processes. No one can perfectly model any process. There are always unidentifiable variables that affect the process. With manufacturing process models, though, we usually identified enough of these variables to be successful. Since I've never met a successful fundamental trader, my conclusions are: 1. Successful fundamental traders must be very rare and that, 2. The "process that drives fundamental commodity prices must be quite complex, and it must have lots of unidentifiable variables or more people would be successfully trading fundamental models.

Have you been trading this year? Why or why not?

I'm strictly a systems researcher—and don't trade at all—it's what I enjoy doing. But it's a two-edged sword. As some bard said, "To trade, or not to trade, your own system: That is the question." In one corner are those who say, "You don't even have enough faith in your system to trade it yourself." I respect that opinion. In our case it's an academic exercise, as we just bought our dream home, which ate 100 percent of our risk capital.

However, there is another corner in the trade/don't-trade-your-own-system room. I've observed that a vast majority of system vendors who trade their own system have overoptimized systems. I've wondered why that so often is the case. It may relate back to the fear/greed problem. I suspect that after three or four losing trades in a row—a perfectly normal event that even the best systems experience—fear sets in. They start thinking, "My system quit. I better start tweaking it a bit." Then they take a perfectly good system, pucker up their lips and give it the overoptimization kiss of death. I don't have those pressures on me. It may explain why I almost never change my system rules.

Are you currently working on any programming projects?

Yes, I'm working on three major projects. The one that is closest to fruition is a new system that will be called "Little Big Horn." It uses a different principle than our other three systems. We use a six-step process to develop a system. The last step after finalizing the system is to paper trade it for six months real-time to see if it continues to behave as designed. It successfully completed its test in June, so it probably will be released for sale in a month or so.

Dave Fox

Education:	B.A., Business Administration, University of Maine (Orono)
Positions Held:	Commodity Trading Advisor, Registered 1984
Favorite Book on Trading:	*New Commodity Trading Systems and Methods*, by Perry Kaufman

Published in December 1999

Nothing perhaps breeds a more clever futures trader than time and money, lots of time and money—and when Dave Fox sold his interest in a family-owned trucking company in 1980, he suddenly had both. Fox is the developer of the successful Dollar Trader program, designed to trade currency futures.

In 1980, retired and only 50, Fox began spending time in the office of a good friend who was both a stock broker and a futures broker. Fox researched and traded a lot on his own also. By 1984, he had realized that in order to trade successfully, one needed a systematic approach. That same year Fox became a Commodity Trading Advisor.

Probably no system is requested more for reports: the system has withstood the test of time; it trades in popular markets; and currencies are trending markets—good for system trading. Its programmer is known for his honesty and openness.

■ ■ ■

Dollar Trader is again in the top ten list. To what, in general, do you attribute its strong performance?

After one designs a trading system, you need a favorable market to generate substantial profits. The first half of this year the European currencies, led by the new Euro, were in a steady decline against the U.S. Dollar, while the Yen was going sideways. This situation has reversed, and the Yen's rise against the dollar has produced a large gain.

What led you to trading systems development? Or, what inspired you to create Dollar Trader?

During the year 1990, the indicators that I was using to trade the USDX and currencies were producing gains commensurate with those of programs that were rated by Futures Truth, so I decided to incorporate those indicators into a software package that could also be rated by your organization.

At that time, testing showed that the USDX was the most reliable instrument, so I requested that my system be rated on it alone.

What do you say to those who say Dollar Trader is difficult to trade?

From my communication with users, the most difficult part is absorbing the draw downs, but if you are going to capture the big winners, you have to expose yourself to losses.

Do you plan any revisions to your systems anytime soon?

In 1996, we expanded the program to be rated on the D-Mark and Yen, filtered by the USDX. In 1999, trading in the D-Mark has almost vanished, and the USDX has been reconfigured to be 57 percent Euro; therefore, we requested that the program be rated on the EuroFX, traded on the CME, and the Yen. One can also trade the Swiss Franc, but I would recommend using the Euro as a filter. Since that is a subjective judgment, it cannot be rated as such. If the Euro continues to be successful, then we should not have to make any more revisions.

Have you been trading for yourself this year? If you have, how has that gone?

There are a number of brokers who trade the program for clients, and my account is traded according to the program by one of them. If I knew of a better way, I would tell my users!

What advice do you have to those who trade your systems or any other system?

The record is of course important, but I think you have to understand and have confidence in the logic to trade a system. I could never trade cycles, because I don't have the confidence that the next cycle is going to match the last one.

Are you currently working on any programming projects?

From time to time our users make suggestions, and I will test any idea to see if it will enhance our results.

What is a common failing of those who are trying to evolve a trading system?

Using so-called continuous contracts will inevitably lead to unreliable results. Any technique that uses a moving average will be corrupted when contracts are "hatched" together. I don't know of any continuous contracts that don't change the closes.

Wayne Griffith

Education:	Mathematics, Physics, and Psychology
Positions Held:	System Development / System Assist Broker
Favorite Books on Trading:	*West of Wall Street*, *Market Wizards*, *TechnicalAnalysis* by Jack Schwager, *Technical Analysis* by John Hill, Sr.

Published in April 2000.

■ ■ ■

Anticipation is again in the top ten list. To what, in general, do you attribute its strong performance?

By "again" I know you are speaking about the "top ten for the last 12 month" list, because Anticipation has been on the "top ten since release" list continuously since having been tracked the requisite 18 months. It will come and go from the "last 12 months" list according to how good or poor the coffee market is, since that is the market you report. I hope I do not sound egotistical when I attribute the strong performance to the trading methodology, and not to the coffee market. Coffee has for the last couple of years shown some life only long enough to get us excited to trade and then dwindles away month after month to a boring nothing.

What led you to trading systems development and what inspired you to create Anticipation?

Poor seat-of-the-pants trading led me to trading systems development. A brief recap: Catching the crash of '87 with OEX put options after only six months of full-time trading for a living led me to think windfall profits were easy. The net percentage gain from the options was windfall even after being cheated out of about two-thirds of the profit via a very, very delayed fill on Tuesday following crash Monday. I immediately switched to futures trading because I never again wanted to experience such a delayed fill or to be told the fill was delayed while in reality someone else pocketed my money.

The first few months of trading stock index futures in the wild, choppy, gappy market which followed the crash was a disaster. I should not have traded in that environment being a novice, should not have traded three contracts at the time, should not have kept them overnight and incurred five digit stop out losses on gap openings the next day, and should not have lost all the money I made on the crash plus some more. How could I do this? How could anyone do this? I did it because I believed regular big profits were immediately ahead. (I

traded small and waited for good opportunities with small risk during my first six months of full-time trading in which I traded the OEX options successfully, but after the crash windfall I immediately developed a mentality of quick big profits.) I did it because I didn't realistically consider the possibility of loss. I did it because I overtraded, risking too much on single trades. I did it because I never stopped to think about money management. At first I thought I did it because of inadequate trading methodology, but later as I moved forward from the novice state I realized I did it because of the absence of money management.

Not yet fully cognizant that my largest problem was money management, I determined to improve my trading methodology. Having spent my lifetime in large scale computerized modeling and simulation, I decided to develop a computerized model which would allow me to see how well a trading procedure had worked in the past before risking money on it in the future. The IBM and compatible personal computers failed to be adequate. I heard about Futures Truth and went to North Carolina to see them. They were nearing completion of a program that would facilitate testing trading procedures, and it was to run on an Apple Macintosh which was far superior in capability to the IBM/compatibles. I asked Mr. Hill if he would sell me a copy of the program. He said he had not had thoughts along that line. Nevertheless, he was kind to me and I took home a program for $2000 that would have taken me probably the better part of a year to develop on my own. I ordered a $13,000 Macintosh system to run it on—Macintosh was more expensive than IBM back then. Inspired by the capable tools I now had, I then spent numerous years, day and night and weekends too, using the Excalibur testing system to investigate the commodities markets. I didn't do anything else for years. I hold the world record for Excalibur testing hours, far surpassing even Futures Truth because they had other activities taking their time, plus they went home nights and weekends.

What inspired me to create Anticipation specifically? This has a two-part answer. First, the technical inspiration. I wanted a system that made a multitude of trades so that it would have a huge historical trade history to provide confidence that the procedure would hold up in the future. I wanted a procedure that overall got more out of various length trends than a typical trend following procedure—it thus had to trade both the trend and the retracements. I wanted it to trade retracements for the additional purpose of being positioned the correct direction quickly in case what looked initially like a retracement was in fact the end of the trend. It had to be quick enough to change from trend to retracement and back that it would function well when the market turned choppy instead of trending. It had to be based on bar chart patterns only (no lagging indicators allowed) to be fast enough to change directions in an anticipatory manner. I should have made provision for a sideways listless market but did not. To match my personality, it could initiate a long position

only when prices were rising and could initiate a short position only when prices were declining. Also, it could not remain long when prices were declining, and it could not remain short when prices were rising—even if it meant that sometimes it would get chopped around switching back and forth with the market direction. This requirement was because I was always fearful that a small move in the wrong direction would turn into a large move. Odd as it may sound, I developed Anticipation emphasizing psychological comfort to trade rather than emphasizing profitability. Lastly, but very instrumental, I attended a Futures Truth seminar toward the end of the '80's in which John Hill said that simple bar charts tell you everything you need to know about a market. He also said you must anticipate the markets. When John Hill speaks, I listen, and those two comments plus Excalibur testing system provided me the basis and tools to develop Anticipation. I named the trading system Anticipation in appreciation of this. Mr. Hill also stated that money management was the largest part of any trading decision—too bad I didn't hear that comment before I squandered my crash money!

The coffee market has a reputation for slippage. How has your experience been with fills and liquidity?

Once I traded five months without any unpleasant slippage. Once, for an entire month, I was robbed on almost every trade, including a limit order executed past the limit, and the damage was so bad I shut down trading. Now, if I encounter unacceptable slippage I find a different floor broker, or enter trades via limit orders and also attempt to exit via limit orders, or bypass trades set up for potentially large slippage.

Does your system require intraday monitoring?

Yes, it requires constant attention. It doesn't make time-management sense to trade it yourself unless you are a full-time trader.

You must be very busy. Does futures trading and programming ever interfere with your personal life?

Personal life? What is a personal life? I have a vague memory that it is a nice thing. I would like to experience it again. Soon. To that end, I am in the process of making changes to the way I approach the futures and systems business. What I have been trying to do is too much for one person, but I've always worked alone to preserve the secrecy of my research results. My personal focus now is trading for a living. For additional income, I registered with the CFTC and NFA as a broker so that I can provide limited system assist support. By limited I mean I will trade for a limited number of clients any or all of the sys-

tems I'm trading for myself, upon proof of subscription and suitability to trade. I don't want to use my time tracking systems I'm not trading for myself.

Do you have any other commentary that you would like to share with the* Futures Truth *readers?

I've put aside my pride and responded to these interview questions fully, discussing not only my strengths and achievements, but also my weaknesses and mistakes. I believe this will help someone—it would have helped me had I read a similar discussion long ago. We small-scale futures traders are in the most difficult arena I've ever experienced, and I wish you all well.

Mike Barna

Education:	B.S., Mathematics, Arizona State University; M.S., Astronautical and Aeronautical Engineering, Stanford University
Positions Held:	President, Trading Systems Design and Analysis
Favorite Book on Trading:	*A Non-Random Walk Down Wall Street*, Andrew W. Lo and Craig MacKinlay

Published in June 2000

■ ■ ■

R-Mesa consistently ranks near the top of our list, to what do you attribute its strong performance?

R-Mesa was developed over S&P intraday data dating back to 1982. In addition, it tests well on international markets like the FTSE and FIB. Further, the technology combines elements of MESA and R-Breaker, both well known and mature. The bottom line with this effort is that it produced what I believe is a robust trading system with known drawdown characteristics that should continue to exhibit superior performance in the future. Of course there is no guarantee that this or any other system will be profitable in the future.

What advice do you have to those who trade yours or any other system?

There are several key issues you need to know when system trading. First, know your system. Know its weak points and strengths. Learn its negative risk characteristics and positive reward characteristics. Understand that the draw down and maximum consecutive losing trades numbers are your numbers. You either designed, bought, or leased these numbers. Understand that these are your numbers and they may sooner or later be reflected in your account. If your account is undercapitalized, relative to these numbers, and you stop trading at the bottom of a draw down then you are selling, if you will, at the exact low of the market move. Likewise, if you start trading at the top of an equity swing high, just prior to a draw down, you may be buying the top of the market, just prior to a swing down in equity. Unfortunately, watching from the sidelines how much money a system made recently entices you to jump in, possibly at the exact intermediate term top in the equity curve. Similarly, learning that your favorite system you follow is in the midst of a large draw down is not very appealing to jump in. By trading a system you are in effect going long the system with a buy and hold mentality. Most system traders I have spoken to know that jumping in and out of the system is counterproductive, yet many still

try to second guess the system with the result of overtrading the system equity curve. If you believe your system is really robust and has a good chance of being successful in the future, plan on adequate capital to trade the system and plan for the inevitable draw down that will surely come. Know your numbers.

Secondly, your trading infrastructure must be perfect. When the system equity curve was created, it was done so without missing a trade, going on vacation, getting sick, having data feed problems, having computer problems, getting missed or bad fills, and so on. The equity curve was created with perfect execution. Likewise your execution needs to be perfect. You can't second guess the system. You can't throw emotional overtones into the execution of the signals. System trading was not meant to be emotionally pleasing. If anything it turns out to be emotionally difficult, but this difficulty can be ameliorated with adequate capital knowing that the potential draw down is well planned for and that human intervention will be needed only if a predetermined level of draw down (risk) is exceeded. If you have a second thought, read filter, for your system, then test that filter and see if it holds up in testing. Your filter may just turn out to be a one time wonder. If you are second guessing the system, you are not system trading but attempting to extract information from the system using it as an adjunct for your discretionary trading, using it as a indicator in effect. This hybrid type of trading is in effect using a buy or sell rule which may be only 35 percent accurate. Clearly using a system as a forecaster or leading indicator is inefficient since the system is meant to be traded as a system and not meshed in with other discretionary tactics.

What do you think the hot markets are this year?

The Grains are showing up on many traders' radar screens as potential intermediate term movers. I also like the Stock Index Futures and the Bonds. System traders are particularly fond of markets that are exhibiting a higher degree of volatility since breakout systems like volatility whereas horizontal markets chop them up.

Are you working on any new projects that you can tell us about?

The research is ongoing continuously creating hundreds of new systems and analytical approaches. Of particular interest now are approaches capturing inefficiencies in the market elements generally overlooked prior to the introduction of newer data mining engines, genetics, neural networks and other nonlinear algorithms. Further, an advance in the integration of the Maximum Entropy Spectrum Analysis into current research has showed promise. John Ehlers and myself have been deeply involved in research into various modeling algorithms incorporated into trading systems. This research includes intermediate term Stock systems as well as short and long term Commodity systems.

What do you think about stock trading systems?

The development of stock trading systems is far more difficult than developing a commodity system since in the commodity world you need to be concerned with just one time series per system. In the stock world, your system needs to work on hundreds of individual time series each one exhibiting unique characteristics. Our work in this area started by evaluating generic trend-following systems that have historically been shown to work quite well over large sets of markets like channel breakout systems. Baselining, stock portfolio systems against buy and hold approaches, gave us performance targets with which to compare the performance of these systems. In the near future, we will see many more stock systems made available to those who wish to trade systems on baskets of stocks. The stock system trading industry pertaining to the individual trader is just now in its infancy.

As a professional TradeStation PowerEditor programmer, what do you think of the security issues with black box systems that are built into TradeStation 2000i?

The bottom line here is that if you have a system that you believe is superior and you are unwilling to subject it to potential security compromise then never give it to anyone, never share the technique with anyone and never sell it. History has shown that just about anything can be cracked so you should never rely on any security layer as impervious. To further protect your technique a DLL programmed for TS 2000I is an option. If you wish to share or sell your system then considering a DLL coded system is critical.

Do you have any other commentary that you would like to share with the* Futures Truth *readers?

By keeping track of our sample or walk-forward testing such as presented here in *Futures Truth*, you have taken the first step toward proper investigation of trading systems you are planning to trade and understanding the system specific drawdown characteristics.

Ziad Chahal

Education:	Political Science, American University of Beirut
Positions Held:	Full-time trading-system developer and vendor (Alfaranda, CTA)
Favorite Book on Trading:	*The Futures Game: Who Wins? Who Loses? Why?*, by R. Teweles, and F. J. Jones

Published in August 2000

■ ■ ■

What do you think the hot markets are this year?

I don't allow myself the luxury of giving similar predictions since similar attempts are invariably wrong. Traders' lives would have been much easier if any of us knew what would be the next trending market. Simply so, nobody does. And that's exactly why even the money-management-gurus can never overemphasize enough the need to diversify through markets, system matrices and money management techniques.

What do you think about stock trading systems?

I still have to see one that actually beats the simple buy-and-hold policy in a nonoptimized long-term test! Trading results are directly related with the involved risks. Since the buy-and-hold system is the one that carries the larger possible risks on a stock-by-stock basis it is an unbeatable system. We can easily come up with trading methodologies that outperform it on a very large number of stocks but *not* the whole stock universe *while* that universe knows only one trading direction, up and then up again. It may be a different story in an extended bear market. But since that has not been the case for the past 20 years or so, and since there is no valid indication that this market characteristic is changing its face anytime soon the best would be to exercise the least in trading-entry/exit judgments!

Here is the best illustration: 90 percent of the billion-dollar stock fund managers constantly fail to beat the S&P500 Index although they also simply buy-and-hold stocks. Obviously so, their assumption that they can outsmart the market by "picking" the best stocks for their portfolio holdings is not helping them either. And that applies for both of those that use technical or fundamental analysis techniques in their stock-picking decisions.

Your BASIS and ARCS systems consistently rank near the top of our list. To what do you attribute the strong performance?

I have a rather shocking answer. I am convinced that a deep valley actually exists between simple technical analysis studies and the methodologies used in successful systems. Technical functions and indicators are mere tools that never are more than 50 percent right. Yet each of these tools has a characterizing personality that can be used by the professional developer to produce a more-or-less robust and solid system. Three years ago, I thought it would be impossible to have long-term successful systems if the methodology is not optimized for each market environment. I owe to the "unknown soldier" his consistent encouragement and the fact that he persuaded me to follow on the track of releasing systems that have a minimum number of parameters and some simple trading rules that can and should be applied on absolutely all markets. I like to think that this is the main factor behind the relative success of my systems although this nonoptimization process does not offer in itself any guarantee of future trading results, specially on short-term basis.

What advice do you have to those who trade yours or any other system?

We first have to admit that systematic trading is a *very* hard proposition since it means committing one's money to ideas that have been studied or developed by the final user, a factor that is bound to hang huge doubts in the trader's mind.

Furthermore, systematic trading also is a relatively dull experience that does not answer to the natural need for that fulfilling adrenaline charge we all look for. In fact, the vast majority of those who use systems have started trading their own discretionary ideas. They experience the losses and the nerve-breaking emotions and finally take the decision to trade systematically. *Yet*, they always seem to be unable or unwilling to let the system work its way without interfering in its signals.

I can bet that a good number of the systems tracked by your magazine are solid and robust enough to be the rival-envy of many professional future fund managers. Nevertheless, these managers constantly make money when our usual clientele loses it. And there can be *only* one reason; they wisely manage their funds when our "systematic" traders are still attempting to manage their emotions.

In short, the decision to trade systematically is an excellent step in the right direction, but the job cannot be completed if you are still looking for ways to beat the markets at the rhythm of 200 percent a year. Try adequate capitalization and unwavering discipline. They make wonders *even* with defective trading systems.

Are you working on any new projects you can tell us about?

I released in March a new system called ONIX that is a frequent position trader for stock-index futures. My newest (July) release though is WEAVER: a happy discovery of an astoundingly robust methodology that works with *one* parameter that is vastly applicable for longer-term trading of all commodity markets while equally offering the possibility of short-term trading the stock-index futures!!!! I hope my work is improving with time. You and my customers will be the judge.

You must be very busy. Does futures trading and programming ever interfere with your personal life?

Well it always does. Right? The truth is it used to interfere more a few years ago when I used to spend 16-hour days in front of my computer testing various trading ideas. I remember times when the absorption was so great I never stepped outside my home-office for a two-month period. Thanks to my loving wife and forgiving kids. Their support was crucial.

Who do you admire in the industry? Why?

I am a fan of John W. Henry who happens to have the largest CTA firm and an enviable consistent track record. I would also like to mention that I have been blessed over the past few years with many friends who have offered me a great deal of much-needed encouragement and support. Of them, I am particularly impressed with a reader of this magazine (whom I won't name) who, besides his unconditional true friendship, has proven to me over the past three years that he has a very lucid mind and practically has system-development running in his veins.

Do you have any advice for those that are new to trading?

Systematic trading is a mid-phase between the common discretionary trading and the allocation of your money to a professional fund manager. I dare assure that less than 1 percent of new traders who decide to trade their own ideas will ever make money. If you do not think you have the material to be in that 1 percent group then look for a trading system that fits your personal preferences and risk tolerance levels. This is a non-forgiving, very tough business. Be prepared for the worst irrespective of any rose-garden scenario a system developer or a broker or a friend has ever painted for you.

Do you have any other commentary you would like to share with Futures Truth readers?

We all may have various points of disagreement and/or private opinions about FT [*Futures Truth*] tracking, system ranking, and general modus operandi.

Yet, we all have to agree though that this is an invaluable institution without which we all would have been much worse be it as future professionals or new or seasoned traders.

FT deserves our support and I hope we can all turn in to make of it a real forum that discusses in-depth all systematic trading problems and features.

John Tolan	
Education:	B.S., Economics, University of Illinois
Positions Held:	Chicago Board of Trade, Merrill Lynch, CQG, Managing Partner of Trendchannel
Favorite Book on Trading:	*Technical Analysis of Stock Trends*, by Edwards and Magee

Steve Marshall	
Education:	M.B.A., University of Denver
Positions Held:	Renizon Corporation, CQG, Managing Partner of Trendchannel
Favorite Book on Trading:	*Reminiscences of a Stock Operator*, by Edward LeFevre

Published in October 2000

■ ■ ■

The Trendchannel system has been ranked in our top ten systems since release date for several months now. To what do you attribute its consistent performance?

[Steve Marshall] The Trendchannel is a robust system. Our original concept for what became the Trendchannel, wasn't a true "system" at all. We wanted to create a simple trend indicator, a tool that would work in any market to tell when the market was trending, the direction of the trend, and consistent support and resistance levels. Somewhere along the development process we discovered its potential as a complete system. The best times to make money are, of course, in trending markets, and we had developed a very stable indicator that tells you when a market is entering a trending phase; and more importantly, when it's sideways and not trending—so you can stay out. Once we had our formula, we tested entering a position when our indicator signaled a new trend and getting out when it signaled a sideways trend. At that point it didn't matter how we varied the parameters, it still made money. Changing parameters affected the number of trades, the length of a trade, and the win/loss ratio, but every combination was potentially profitable. So, because the system is so robust, it has worked as well in real-time as it did in historical testing.

[John Tolan] The Trendchannel formula is different from all the other technical studies out there. We developed our system by correcting what we found to be flaws or drawbacks with the other previously available indicators. Our historical testing helped us to further improve our formula design. It is

however, not "optimized." We use the same formula for all markets and it works well regardless of parameter settings. Our goal was to create an automatic trend indicator that is easy to use—green light, red light, even though the formula behind the bottom-line signals is more complex. The Trendchannel formula combines adaptive measurement of volatility, momentum, and both long- and short-term trend direction. The system defines trends in a way that is consistent with classical trend analysis, but does a better job keeping you out of sideways (trendless) markets and keeping you in major trends.

Do you have any advice to those who trade your system or any other system?

[Steve Marshall] Once you have researched and chosen a system to follow, trade that system's rules precisely, and give it time to work. Know the system and believe in it. Make sure that you understand how the developer intended the system to be traded and try to do exactly that. If you can't execute trades the way the system requires, don't expect to duplicate the system's performance. If you are unable to follow a system exactly, many brokerage firms have a system assist division that will trade your account based on the rules of the system. Once you're following a system, give the trades some time to work. A system with a long-term track record with good results may still have periods of months with no profits. You have to be ready to accept that and stick with the system long enough to see it work.

[John Tolan] On our Website we offer a free trading guide called "The Simple Rules for Successful Trading." The information in the guide is good advice for any trader. If you can't apply these rules to our system or any other system, then you will not be successful trading.

Is there anything that you would tell to those new to trading commodities or stocks that you wish someone had told you when you were getting started?

[Steve Marshall] Have a plan before you start trading. Plan out everything you can. Know when and why you will buy and sell. Know which markets you will trade. When you're starting, it may be enough to trade in just a handful of markets. Know how much you are willing to risk—in each trade and in your account overall. The more rules you have, the easier it will be to make trading decisions. Also, realize that you shouldn't trade just to be in the market. Nobody has a system that makes money under any market condition. There are times you should be out of the market waiting for a signal. Have patience.

[John Tolan] Oh yeah. . . . That was my initial motivation to write the Trendchannel trading guide—to share this advice. The vast majority of those who begin to trade lose money. That does not happen by chance. It's predictable because they make and repeat the same basic mistakes over and over again. If you can use a trading system with rules that enable you to avoid making these mistakes, you will be successful.

What do you think the hot markets are this year?

[Steve Marshall] Crude Oil, Natural Gas, and the Euro Currency have been hot markets for us this year. But our approach is trend *following*. We don't try to predict anything. We do what the market tells us to do. We go with all trend signals and see what happens. I have no idea of expectation of where the market is going to go, but no matter what it does, I know what I will do, how I will trade it. That's the security you get trading a purely mechanical system that you have confidence in.

[John Tolan] The high volume markets will most likely be the "hot markets." In other words, where there is volume there will be trends. In addition to price, we track the trends in volume. That's one of the methods we use to select markets to include in our stock and futures portfolios.

How do you prepare for your trading day?

[Steve Marshall] Well, for our futures account, we have a system-assist broker trading the Trendchannel system for us. They execute trades perfectly, plus it is creating an actual trading track record, and frees us from watching the markets. We do our own stock trading based on the end-of-day signals from our Trendchannel Stock Report. I spend less that 30 minutes a day on trading. Once our reports have been generated, I place the trades for the next day's open. I might check the market once during the day, but generally, I just wait for our report to come out each afternoon.

Do you think there is a future in technical stock trading systems?

[John Tolan] Yes, for all of the same reasons we believe there is a future in trading systems for futures. Most beginning traders, including stock traders, try to figure out which way the markets are headed by looking at charts, analyzing technical indicators, or by reading news and fundamental information. When these approaches fail, they seek other methods or advice. One solution they will find is to trade an automatic system with a proven track record. And, of course, once they figure that out, and find a system, they will also seek help in tracking and entering the trades for the system. That's why we also believe there is a big future in system assist services for stocks as well as futures.

[Steve Marshall] Absolutely. . . . We have many subscribers, including brokers, who use the Trendchannel trading system for stocks. The Trendchannel works just as well with stocks as it does with futures.

Do you consider them (stock systems) a viable way to trade stocks successfully?

[Steve Marshall] Yes, no question about it. The volume in stocks can create very strong long-term trends. A purely technical trader can do very well in the stock market.

[John Tolan] I believe it's the best way. On our Trendchannel Stock Report, we screen out the 50 most actively traded stocks, the Trendchannel 50. These high-volume stocks have produced some spectacular trends since we started tracking these two years ago. And of course the Trendchannel Trading System does well when markets have big trends.

Are you guys working on any new projects you can tell us about?

[John Tolan] We just released the Trendchannel Software for Tradestation. I've always thought it would be great to have a trading system that works just like a traffic signal, with automatic "green light, red light" indicators that instantly flash on the charts when to buy and when to sell. That's exactly what the Trendchannel Software does. We are making it available so that anyone can trade the Trendchannel system in any market, in real time, and in any time frame they choose. You can totally automate your trading.

John Ehlers

Education:	B.S.E.E., and M.S.E.E., University of Missouri, Doctoral work at George Washington University
Positions Held:	Electrical Engineer, specializing in Information Theory; President, MESA Software
Favorite Book on Trading:	*Trading Systems and Methods* (3rd ed.), by Perry Kaufman

Published in December 2000

■ ■ ■

R-MESA consistently comes up as one of the top S&P and daytrade systems. To what to do you attribute its consistent performance?

In a word, robustness. R-MESA uses R-Breaker by Rick Saidenberg, one of the Futures Truth top ten systems of all time, as its origin. Mike Barna and I combined R-Breaker with the Mesa cycles-measuring algorithm to make the trading signals adaptive to current market conditions. MESA makes it easier to buy at the bottom of the cycle and easier to sell at the top of the cycle. Combining a proven trading system with theoretical considerations was just the beginning. The robustness arises from the extensive out-of-sample testing we have performed on data going back to the beginning of the S&P contract. We know the system will perform well in the future because it has been tested against all kinds of market conditions in the past.

Do you have any advice to those who trade your system or any other system?

The biggest error I see traders make is the inability to stick with a trading system through the draw downs. This can be due to a variety of considerations, ranging from undercapitalization to just plain fear. I find this strange, because the major reason for trading systematically is to have the confidence that the system will perform in the future much like it has traded in the past. It is not difficult to scan past performance for the maximum draw down, so the required capitalization can be estimated. It is a virtual guarantee that this maximum draw down will be experienced again, particularly in nonrange-bounded markets like the S&P, and a trader should be prepared to withstand it. I can understand the fear factor where traders are using a black box system developed by a vendor. It is important that the vendor show both the long term and short term historical performance to demonstrate system robustness and that the system has not

failed in recent market activity. This whole issue of trust is where Futures Truth comes in. By acting as an independent third party, the trader can have confidence that the system has been exercised by knowledgeable traders and that your unbiased evaluation will establish performance credibility.

Is there anything that you would tell to those new to trading commodities or stocks that you wish someone had told you when you were getting started?

My biggest failure as a trader is being married to a trade. Once I have made a decision to enter a position, I absolutely know I have made the correct decision. When the trade goes against me a lot, then I shift my thinking by saying to myself that I cannot afford to get out now and I will stick it out until the next cycle comes along. This simply means that I lack the discipline to be a discretionary trader. I wish the pitfalls of discretionary trading had been pointed out to me earlier so I could have bypassed the educational expense. I learned a lot, but am now focused on systematic trading.

What do you think the hot markets are this year?

I judge performance parametrically. For example, I look at the profit per trade and the MAR Index (net profit divided by draw down). As a small trader, I also look at the return on margin. Having said that, I conclude the oldies are the goodies. As you have pointed out, R-MESA continues to perform well in the uncertain S&P markets this year. I see some movement to the E-Mini because the slippage can be minimized. I also trade the Treasury Bonds.

How do you prepare for your trading day?

Believe it or not—nothing. My work has already been done in developing the systems. The entry points are in place and the stops are established. There is nothing left for me to do. I have supreme confidence the systems will perform. I just keep score at the end of the day.

Do you think there is a future in technical stock trading systems?

Absolutely! However, trading stocks is vastly different from trading futures. I see stock trading as a two-step process. First of all, with stocks you have to know what to trade. You have already made that decision with futures. Only after you have lined up some likely candidates, can you implement a trading system. In this sense, the stock trading system is a market timer. Since most stock traders trade the long side only, the most successful systems will be something primarily designed to tell them when to exit the trade. A parabolic could work well in this scenario. Also, channel breakout systems tend to be relatively robust for stocks.

Do you consider them (stock systems) a viable way to trade stocks successfully?

Given that people have now found that stocks can and do decline in price, I think it is the *only* way the small investor can successfully trade stocks. For example, a system will enforce the discipline to get out of a stock position when the price declines. As I said, that is my own trading weakness, and I suspect is a weakness in many traders. All the other benefits of systematic trading also apply. For example, it might cost $8 commission to buy 100 shares and another $8 to sell. That means, with slippage, that the cost per round turn will be about $0.30 per share. When we test our system, we know we must make at least $0.30 per share profit, on the average, just to breakeven. That is more difficult than it sounds. There is no margin leverage with stocks. This means making breakeven is more difficult, even with a successful trading strategy, and the profit return on account equity will be smaller than with futures. On the other hand, not being margined means that trading with a system can produce lower draw downs.

Are you working on any new projects that you can tell us about?

I have written a new book *Rocket Science for Traders* to be published by John Wiley & Sons next spring. The theme of the book is introducing modern digital signal processing techniques to technical analysis. As a result, several novel and unique indicators have been invented—some that could not be programmed without the new processing techniques. I am forever in a research mode. In the course of writing the book, I have started to investigate applying nonlinear filters to match the waveshapes that result from the probability density function of nonstationary market prices. This is fascinating stuff, with some preliminary successes in creating indicators. My general approach is to develop concepts on theoretical waveforms and then transition to real world market data. The ultimate proof of performance will be the development of profitable automatic trading systems based on these nonlinear filters.

Who do you admire in the industry? Why?

There are a lot of really great people involved with trading. I admire them in a variety of categories: as businessmen, as traders, and as technicians. When we narrow the industry down to system developers, I have great respect for Bill Brower and Nelson Freeburg for their thoughtful application of first principles. I admire Mike Barna for his vast knowledge of what works and what doesn't work as well as a work ethic that is second to none. Mike and I have collaborated on a number of projects. We work well together because we are kind of a combination of Steinmetz and Edison in our approaches. I like to think about the theoretical aspects and Mike will try a jillion things until something works. That's a pretty powerful combination.

What led you to trading system development?

Necessity. I have traded my own accounts discretionarily and have been successful. On the other hand, I have had some relatively large declines in equity. I realized that I never knew when my hand went cold, even though I was trading the same way as when I was successful. When I started system development based on my own discretionary rules I found that, indeed I was getting wild swings in my equity in my hypotheticals. I also found that I was using some rules that were not objective and could not be programmed into a system. By carefully crafting the systems and watching key parameters such as profit per trade, the MAR index, and the profit factor, I found that I could bound the excessive swings. I now could reasonably estimate what I could expect in the way of profit and draw down.

You must be very busy. Does futures trading and programming ever interfere with your personal life?

Of the 16 waking hours per day we all have, the majority of that time is spent at work. Because of that and because I feel a person is defined by his contribution to society, my priorities are heavily weighted toward work. Besides that, this is fun! How many different ways can one be creative and see the success of that work pay off directly in dollars as well as acknowledgment by one's peers?

Do you have any other commentary you would like to share with our readers?

I would like to thank *Futures Truth* for providing the service of independent and objective evaluations of commercially provided trading systems. By being exercised by knowledgeable traders like yourselves, your readers can have confidence in their selection when it comes time to purchase a system.

"Chuck" Charles LeBeau	
Education:	Degree in Finance from California State University, Long Beach
Positions Held:	1967–1988 Vice President, Regional Futures Director, E. F. Hutton; President—Island View Financial Groups; President—StreakingStocks.com; Founder of System Traders Club; Co-author of *Computer Analysis of the Futures Market* (with David Lucas)
Favorite Books on Trading:	*Computer Analysis of the Futures Market*, by Chuck Le Beau and David Lucas *The Ultimate Trading Guide*, by John Hill, George Pruitt, and Lundy Hill *Trade Your Way to Financial Freedom*, by Van K. Tharpe
Published February 2001	

■ ■ ■

Sidewinder consistently comes up as one of the top Bond systems and as one of the top ten systems since release date. To what to do you attribute its consistent performance?

This is a good example of having a variety of systems each of which is designed to trade a particular phase of a market. The Sidewinder system was designed to be active and to trade countertrend when the bond market is in a broad trading range. It just so happens that the current bond market fits the characteristics the system was designed to exploit. When the bond market stops trading sideways, the Sidewinder system should shut itself down and one of our trend following systems should start trading actively and perhaps do equally as well. As much as I like Sidewinder, I wouldn't want it to be my only system for trading bonds.

Do you have any advice to those who trade any of your systems?

Watch out for the S&P market these days. I think there is more volatility in that market than we had expected when we originally designed the S&P systems. The high volatility could be a problem.

Is there anything that you would tell to those new to trading commodities or stocks that you wish someone had told you when you were getting started?

Remember that we can always control our losses very precisely but it is hard to control profits. Draw downs and losing periods are usually caused by a reduc-

tion in the size of the profitable trades rather than a big string of sequential losers. Be sure to make every effort to maximize the size of your profitable trades by concentrating on timely exits.

What do you think the hot markets and stocks are this year?

The stock market will probably continue to be the hot market for the foreseeable future. I like any stock that has high liquidity, high volatility, and that displays a rapidly rising ADX after coming out of a basing pattern. We list those stocks every day on our StreakingStocks Website.

How do you prepare for your trading day?

Whether I am trading stocks or futures, I prepare the night before after the markets are closed. I rarely make any intraday trading decisions. I try to know exactly what I am going to do and I enter all my orders the night before. Then I can sleep late and go out and play golf or work on writing a book or article without worrying about the markets.

Are you working on any new projects you can tell us about?

Well, I have gotten very involved in doing research on stock trading in recent years. Some of the strategies I designed for futures trading seem to work even better in stocks. I have the new www.StreakingStocks.com Website up and running and I'm going to start a stock hedge fund later this year. I'm also working on a new book about exits and trying to do a revised edition of the Computer Analysis book. I'm also doing some teaching in my workshops as well as private consulting. All of it is keeping me very busy.

Who do you admire in the industry? Why?

I admire Bill Eckhardt who helped train the "turtles" for Richard Dennis and who is a very successful CTA on his own. I think he really understands trading and how to combine good strategies with sound money management. I wish he would write more articles or a book.

What led you to trading systems development?

I was a good discretionary trader who got lazy. It's a lot easier just to program your knowledge into the computer and let the computer do the analysis and all the hard work. When I was much younger I used to get a big thrill out of trading and making all those critical trading decisions on a minute-by-minute basis. Now I no longer get a rush when I'm trading. I would rather be out playing golf.

Between the Systems Trader's Club, StreakingStocks.com, and your commodity systems, you must stay very busy. Does futures trading and programming ever interfere with your personal life?

No, it doesn't interfere at all. On the contrary, I feel that since I became a systematic trader I actually have a life. I now have time to enjoy my grandkids and work on my golf game instead of watching a computer screen all day.

Do you have any other comments you would like to share with our readers?

Yes, don't be afraid to take losses. I think a good attitude about losses is what makes a successful trader. Losses are simply an unavoidable cost of doing business. Also, be sure to monitor your systems on a regular basis and look for weak spots. Too many traders think they only have a problem once their equity is down 50 percent or more so they don't monitor their systems until the losses get their attention. By then it is probably too late to do anything constructive other than to stop trading before you lose it all. Monitor your systems even when they are winning. There is always room for improvement.

What do you think is the future in technical stock trading systems?

I am amazed at what I see going on in the stock market. I regularly see stocks moving 15 percent to 50 percent or more in a day. It will take a very good technician to profit from those short-term moves and the buy-and-hold folks are going to be in for a terribly rough ride. I think that technical analysis is going to become more important and more popular than ever.

Do you consider stock systems a viable way to trade stocks successfully?

Yes, no doubt about it. What I like about stock trading is that there are so many stocks we can watch. I can set up some very stringent criteria and if I look at 9000 stocks I can always find something that is doing exactly what I want. The big problem I have is the close correlation of all the stocks with the general market. If my timing is good on an individual stock but I mis-time the direction of the market I am going to lose money nine times out of ten. Timing the general direction of the market over the short run is very difficult but I think it can be done. I'm working on it.

If you have a preference, would you prefer to trade commodities or stocks on a mechanical trading system? Why?

I have no real preference and I'm willing to go where the opportunities are. I think it is a mistake to approach decisions with an either/or attitude. Most often, "all of the above" is the best answer. Right now, I like the opportunities in the stock market but I would still be watching for opportunities in futures.

Lundy Hill

Education:	B.S., Electrical and Computer Engineering, Clemson University
Positions Held:	Ex-NASA "Rocket Scientist" Engineer; Three years in the Pits of the Chicago Exchanges; Current full-time trader, CTA, Trading Systems Developer
Favorite Books on Trading:	*Market Wizard I and II*, John Hill's early (and late) stuff, *The Disciplined Trader*, Darvis' books

■ ■ ■

We consider the Stafford S&P Daytrade one of the most consistent systems out there today, and it has tested extremely well on the NASDAQ. To what to do you attribute its consistent performance?

Our Volatility Based Daytrade system is just what its name says. It uses the actual volatility of the S&P 500 market to dictate buys, sells, and exits. So it changes. I think stock index volatility is here to stay. Traders must use that in their plan of attack.

Do you have any advice to those who trade any of your systems?

Not only my systems, but any systems. Do your statistical studies. Look at the past performance. Calculate things like the average monthly profit, average daily profit, average profit and loss, and worst-case expected draw down (a rule of thumb says this is the yearly standard deviation). Then you know what to realistically expect when trading.

Is there anything that you would tell to those new to trading commodities or stocks that you wish someone had told you when you were getting started?

Yes. Any trade can have only four possible outcomes. A large profit, small profit, small loss, and a large loss. Three of those are acceptable.

What do you think the hot markets and stocks are this year?

Stock Indexes. Stock Indexes. And *stock indexes*.

How do you prepare for your trading day?

First I analyze my trades for today. Where I was right. Where I can improve. Then I look at the market from a top-down approach. What do I expect long

term, short term, and tomorrow. I try to have some idea of what may happen today. Key levels. But, I don't blindly stick to this perception. If the market proves me wrong, I will try to adapt.

Are you working on any new projects that you can tell us about?

We have new NASDAQ systems coming online. But, my immediate goal is the successful application and blending of all of our approaches. I think we have the technology *right now*! But, it is another matter to successfully apply it. I want an overall portfolio approach of systems trading on stocks, futures and indexes. Long-, intermediate-, and short-term time frames.

Who do you admire in the industry? Why?

I think that should be obvious. Don't you?

What led you to trading systems development?

It seemed the next logical step given my background and my family's background.

Does futures trading and programming ever interfere with your personal life?

Sure. You should see Thanksgiving dinner at the Hill house. Five traders around the table. Sure, we are going to talk trading, but, I have other passions in life I pursue with equal vigor. I think that is important to stay fresh in your trading.

Do you have any other comments you would like to share with our readers?

Most of the clichés you hear about trading are true. For instance "limit your losses." If your average loss is $500, try using a $350 to $400 stop loss instead. If your average profit is $500, work on improving your trade entry and profit objective calculations and try to get your average profit to say $600 to $700. Do those two (simple?) things, and I think you will be amazed at the results.

What do you think the future looks like for technical stock trading systems?

Great. I think the increase in volatility of individual stocks will lend itself to a systematized approach to trading stocks.

If you have a preference, would you prefer to trade commodities or stocks on a mechanical trading system? Why?

I think diversification is one of the tenets of successful trading. Therefore, as I said before, trade and invest in both, and across multiple time frames.

Peter Aan

Education:	B.M., M.M., University of North Texas
Positions Held:	I've been a professional musician since I was 15 (I'm 51 now), and I have performed with Symphonies, Operas, Rodeos, and Elvis! CTA, PWA Futures; Broker, Dillion Gage, Inc.
Favorite Book on Trading:	*New Concepts in Technical Trading Systems*, by Welles Wilder

Published in June 2001

■ ■ ■

All of your systems have shown consistency. However, the last few years have been tough for trend followers. To what do you attribute consistent performance and why have trend followers suffered through the last two or three years?

It has indeed been a rough road for trend followers in recent years, myself included. Even the legendary Richard Dennis called it quits (again), and the *Wall Street Journal* ran a story on the toll that the markets were taking on prominent CTAs. When you analyze it, we've seen the markets do this many times before. What was different this time was the persistent nature of the choppiness. When my programs started into a draw down, I was salivating just a few months later, thinking that the markets were due to go into another trending phase soon. Little did I know that we had many more months of rough water. We have had spurts of trending here and there, and my systems have caught whatever trends were there, but it's been a while since we've had what one could call a "normal" year.

Do you have any advice for those who trade your systems or any other systems?

There are three keys to success for any trader, regardless of methodology.

1. Have a trading plan.
2. Have the discipline to follow that plan.
3. Use good money management to help survive the inevitable losing periods.

That's it. Of the 95 percent or so of people who lose money, I suspect that virtually every one of them violated one or more of these tenets. Following

these three rules won't guarantee you a spot in the winner's circle, but your chance is pretty close to zero if you violate them.

On a more specific basis, before you begin to trade a system, you should examine where it currently is in its equity curve, both on a portfolio basis, and an individual market basis. I feel better starting to trade a system after its been doing poorly.

Is there anything that you would tell to those new to trading that you wish someone had told you when you were getting started?

The above three rules are good advice for anyone who hasn't yet learned them, novice or not. One of the first systems I learned and admired in the early '70s was Donchian's Weekly Rule. Sometimes I wish someone had forced me to trade that system and no others! And think of all the money I would have saved on computers and software! Also, a subscription to *Futures Truth* magazine is a must for any trader who purchased trading systems. I wish FT had been around during my early years.

What do you think the hot markets and stocks are this year?

Coffee, Yen, Franc, and Euro Currency. I'm guessing, of course, and I name these only because the currencies and coffee have been the trending champs over the decades.

How do you prepare for your trading day?

I download data every evening and run my systems on TradeStation. My orders are typically entered before the opening, and then I am mostly through for the day.

Do you think there is a future in technical stock trading systems?

This is not an area in which I have done much work. I suspect that it would be difficult to devise mechanical systems that can match in real-time the long term return of about 11 to 12 percent that stocks have returned over many decades. Many mutual fund managers have underperformed the market, so one must assume that beating the market is not an easy task.

Do you consider them (stock systems) a viable way to trade stocks successfully?

I don't care much for trading stocks, except perhaps for entertainment, but I do favor holding equities for the long-term. Unlike most futures markets, stocks have an organic upward bias to them. Long-term investments in equities take advantage of that bias.

Coming from a system developer, this may seem like heresy, but here is my long-term "system" for making money in equities.

a. Make monthly purchases of mutual funds or exchange traded index funds, preferably in a tax-advantage vehicle, such as a 401k. Dollar Cost Averaging is the eighth wonder of the world.
b. Diversify among different market segments (large cap, small cap, growth, value, foreign).
c. Rebalance the portfolio as needed.
d. Avoid bonds and individual stocks.
e. Don't even think about timing the market. The market won't do anything over the next 30 days that will make much difference 20 years from now.
f. Pray for frequent bear markets. (If you're still buying equities, why would you want them to go up?)

Are you working on any new projects that you can tell us about?

I don't do nearly as much research as I did in previous years. Perhaps it would be a good exercise to "wipe the slate clean" and research some area of technical analysis that is dramatically different from what I have done previously.

Who do you admire in the industry? Why?

I've always admired Larry Williams for his creativity, both in system design and system promotion. Larry never bores me.

What led you to trading systems development?

I believe that my personality led me naturally to system trading. From the beginning, I have always favored approaches that are mechanical. To come into the office every morning and make trading decisions on every market would be torture for me. For others, trading a system verbatim would be torture.

You must be very busy. Does futures trading and programming ever interfere with your personal life?

What personal life? Seriously, since I don't do a lot of research anymore, I put in a normal 60 hours a week, just like every other American. For entertainment, my favorite pastime is going to the movies.

Do you have any other commentary you would like to share with* Futures Truth *readers?

Don't buy a trading system from the vendor who promises the most.

Michael Chisholm	
Education:	B.S., cum laude, Psychology, University of Maryland
Positions Held:	Technical Engineering Writer on Polaris Submarine Missile Program Trader since 1959; Publish advisory since 1976; President, Taurus Corporation
Favorite Book on Trading: *Published in August of 2001*	*Trading For a Living*, by Dr. Alexander Elder

■ ■ ■

The last few years have been tough for trend followers. To what do you attribute your Grand Cayman System's consistent performance and why have trend followers suffered through the last two or three years?

Most market analysts see the past several years of market behavior as extraordinary, and in some aspects they are. I have seen many similar periods of choppiness at different times over the last three decades, but recent years have been different in that the relative lack of trendiness has lasted longer than in the past, and in many cases the markets have at the same time been more volatile than usual. I believe this is a result of the topping out of the economy, changes in the worldwide economic situation, and ever-increasing investor uncertainty.

The Grand Cayman System has weathered this period well, and I believe that is due to its long term approach of being able to catch what trends there are and ride them successfully. Also, the fact that the system is not overoptimized has helped it while other, heavily optimized systems, have failed due to the schizoid markets.

Do you have any advice for those who trade your systems or any other system?

The primary elements in being successful at system trading are the system itself, money management, diversification, and psychology.

First, and foremost, you need a system that has proven itself over a sufficient period of time, at least 5 to 10 years, and is not overoptimized. Subscribing to *Futures Truth* is a good way to select such a system.

Next, you need to make certain you are well-enough capitalized to trade whichever portfolio of commodities you choose, betting that whatever the maximum draw down has been in the past, it will be exceeded at some time in the future.

To me, diversification between commodity complexes is important so that if one group of commodities goes "awry," the odds are the other companies will make up for it.

Perhaps, most important is psychology, which entails having the discipline to follow through on all the system's recommendations precisely as given and the patience to wait for new signals and stay in trades until the system itself calls for an exit.

Is there anything that you would tell to those new to trading that you wish someone had told you when you were getting started?

That trading futures can be easy and profitable, but only if one follows the general guides I just gave. If one is into developing his or her own systems, then it is hard, time-consuming work—unless you love research like I do, then it becomes fun.

I think the psychology of trading may be the most important element, since trading exposes one to all the emotions, from joy to fear, from exhilaration to grief, and the trader must be prepared for all this and be able to deal with it appropriately.

What do you think the hot markets and stocks are this year?

I'm anticipating some major bull markets in the Grains and Precious Metals this year, and the Currencies cyclically are due for significant upmoves. Given the deflationary environment coupled with building inflationary pressures, predicting price movements for the coming months is more difficult than usual.

How do you prepare for your trading day?

We get our download just after dinner, and my partner (wife) and I do the analysis on TradeStation then and place our orders in the morning. Most of the rest of our time is spent doing research, keeping four computers running tests on TradeStation almost 24/7 looking for improvements in technique.

What is your preference for style of trading? Do you prefer day, short term, intermediate term, or long term trading the best?

For me, it's a question of profitability and personality. I don't enjoy day trading, so I don't do much of it, profitable though it can be. My own preferences are either very short term trading or very long term. I believe they can be the most profitable, and they fit my style and personality best.

You mentioned very short term and long term trading as your favorites. Quite a disparity here, isn't there?

Well, yes and no. Maybe I'm manic-depressive [smile], but I go through periods of time when I want the excitement of very short term trading, and then

other periods, especially around vacation times, when the long term trading fits my lifestyle and personality at that time best.

Your Grand Cayman System falls under that long term category. Overall, do you feel that kind of approach is superior to other types of trading like day trading and short term?

Yes, looking at making money in the futures market more as an investor than as a trader, long term, and even very long term trading is probably the best investment in the world today, assuming of course, you have a valid long term trading system.

Are you working on any new projects that you can tell us about?

Yes, an exciting new long term system we're almost finished developing called Horizon, which Rachel is programming as a stand-alone windows CD-ROM. I'm working on new versions of my original book, *The Taurus Method*, as well as my book on the psychology of trading, *Games Investors Play*. And, just in case that isn't enough, I'm also outlining a book on long term trading while both of us continue running research.

Who do you admire in the industry? Why?

This, I know, is nepotism, but it's my partner and wife Rachel who in five years as an AP and computer programmer (and web master) is a work of art!

What led you to trading systems development?

This could be an entire book, but for brevity it was helping my father develop systems to bet on horses, which was the foundation of my development of system analysis. Incidentally, he did pretty good at the track with his system!

You must be very busy. Does futures trading and programming ever interfere with your personal life?

Trading commodities is our life, corny as that may sound. We do things recreationally of course, but we find our business work not work at all, but play.

Do you have any other commentary you would like to share with* Futures Truth *readers?

Yes. Know yourself. Without the right psychological perspective, no matter how sound your system, you cannot succeed for long.

Michael A. Mermer

Education:	B.A., J.D., Hofstra University
Positions Held:	1990 to date, President, TradersSoftware.com CTA
Favorite Book on Trading:	*Technical Traders Guide to Computer Analysis of the Futures Market*, Lebeau and Luca

Published in October/November 2001

■ ■ ■

To what do you attribute your ETS Daytrade System's consistent performance and why have trend followers suffered through the last two or three years?

We do not totally agree with the statement "tough" for trend followers. We have found the contrary and ETS has had some of its best years ever the last few years. Many newer systems were designed and optimized for bull markets only. The systems fell apart once the market started to decline. This last year, we have been in a bear market on most major indexes so the system's design had to take into account what would happen in bear markets. Since ETS was originally designed in 1990, during volatile up and down markets, it has held up very well over time, especially the last few years.

Do you have any advice for those who trade your systems or any other system?

Trends are defined by the time frame you look at. On a one-minute chart or tick chart, there are dozens of trends per day to trade. As you go up in time frame, the number of trends decreases. You need to find a comfortable time frame to trade that meets with your personality. What works for one trader may not suit another and the number of trades you wish to make per day should determine the time frame you look at. As a general rule, the longer the time frame, the more relevant the data is and more reliable the signals—this goes for most any trading system.

Is there anything that you would tell to those new to trading that you wish someone had told you when you were getting started?

Yes. Expect both wins and losses. If you are looking for a system that has no loser, forget trading and go into another hobby. Systems are best judged by the risk/reward ratio, not the number of wins to losses. You can make a tomorrow of money on a system that is only right 40 percent of the time so long as you have big wins and small losses. The gross number of wins is meaningless with-

out looking at an entire system. And, trading is not for everyone. Don't be afraid to quit if you don't like it. If after six months you find you cannot make money consistently, don't even consider making it into a business.

What do you think the hot markets and stocks are this year?

S&P, NASDAQ, and all the major market indexes. I think we have a very volatile big year ahead.

How do you prepare for your trading day?

No special warm up, but start the day knowing you are going to follow your trading system rules. Rule #1—Do the right thing during the *day* so you don't have to stay up all *night* worrying about the mistakes you made during the day.

Do you think there is a future in technical stock trading systems?

The markets are traded technically so there is no difference in a stock system versus a futures system. As far as development is concerned, I think, most every algorithm has been tried and developed but there is a possibility someone could come up with a new one. As far as popularity is concerned, the markets are traded technically and trading systems will always be around. Practically no one can beat the odds of a mechanical trading system—even a mediocre one.

Do you consider them (stock systems) a viable way to trade stocks successfully?

Yes, a system that works on futures should work on stocks because a chart is a chart is a chart! The system does not care what it is being applied to.

Are you working on any new projects that you can tell us about?

Yes, we just released our like Java Signals, which is our ETS Signals on very fast tick charts. It would be impossible for us to send out fast day trading signals manually so we have automated ETS into a Java version and made it available to all those on the Web. There is a free trial at www.traderssoftware.com. Also using the latest technology, we have developed a Live Squawk Box for the S&P Futures at Realtimefutures.com. The squawk box is totally live and an invaluable trading tool.

Who do you admire in the industry? Why?

This is a hard question. No one in particular except other advisors like myself that have the guts to pull the trigger on signals that are traded by hundreds of traders. We know that you can't judge a system by its performance on any one day, but we know that if traded consistently over time, our systems outperform and make money.

What led you to trading systems development?

My basic interest in the markets, my technical computer skills, and in 1990 when TradeStation became available—because it allowed someone like myself who was not a programmer by trade to automate my trading research and discoveries.

You must be very busy. Does futures trading and programming ever interfere with your personal life?

I try not to let this happen. Too much of anything is not good and this is true of trading as well. Leave yourself time for leisure.

Do you have any other commentary you would like to share with* Futures Truth *readers?

Yes, Futures Truth results are great for traders to look at. But, make sure you also like the strategies used in the systems you choose to trade. Just because a system tests well and is ranked #1 doesn't mean you will like it. Always find out what type of methodology is used in the system and make sure it is consistent with your personality and overall risk tolerance.

A TALK WITH LARRY WILLIAMS

by Rob Keener

Larry Williams is a seasoned trader with over 32 years of experience. He has authored several best-selling trading books. Mr. Williams also won the Robbins World Cup Trading Championship by actually trading $10,000 to $1,100,000 in one year. He is a frequent speaker at seminars where he places real-time trades in front of the audience and gives away the profits!

How do you prepare for your trading day? Understanding that trading is a strenuous day-to-day business, is there any ritual?

I don't think it's all that stressful frankly as long as I know where my stops and my entries are. I know the worst thing that can happen and I've already accepted that can take place and probably will. So where's the stress? I think it's something we induce in ourselves by worrying about every tick and price. I figure I'm not going to worry, it's just going to happen.

So, you don't do Yoga?

No, I come to the markets with a pretty basic approach. A lot of commodity traders get beat up on occasion, not rare occasions, but frequently. So, I just have accepted that, I think it's the acceptance of it that gets rid of the stress.

Do most people create failure in their trading strategies? Is it that they don't just go ahead and accept the risk and understand that there is stress involved?

Yes! They don't place stops. So of course they're stressed because they don't know what's going to happen to them.

So they are taking the first profit and letting the losers run?

Yes or they don't even have a target or a stop. They don't know what to do. They don't have any rules. Any situation whether it's life or it's the markets, if you don't have any rules, you're going to create insanity real quick.

Do you think it's possible to use a pure systematic approach without any discretion at all?

Sure it's possible and I've gone through that where I've traded purely systematically. I think that on occasion you can make a judgment call because the future is never like the past and all of our systems are based on the past and things do change. There is a point where you can use some discretion in the market not a lot. If you need to make a judgment call, then that's ok to do that, just don't do it everyday because then you don't have a system.

Are you constantly developing systems or do you have four or five concepts that you follow?

I've got about four concepts. I just keep going down those same tunnels looking for some cheese. My mind is in one mode. I keep mining the same approaches.

Do you think there is an advantage in diversification? (You can diversify over time, markets and strategies.)

I don't like to diversify. If you have a trend-following system that catches trends, you probably need some diversification. I just trade S&P's and bonds. That's it, nothing else. I have some stuff that's held up pretty well after a long time period so I'm just going to trade that. My only diversification would be various entry techniques.

Where do you think it's all going? I talked to a floor trader that trades in the S & P pit for a bank and he was concerned, a lot of guys in Chicago are worried that the pits are not going to be around forever.

When you look across the world you see the Sydney Futures Exchange has gone electronic, it's happened in Europe; my prediction is that the Chicago Board of Trade is going to become a very nice shopping center. There will be some warehouse full of computers and that will be the future of the commodity exchange.

So you have four or five different basic systems?

I use approaches. I use volatility break out. I use pattern recognition. I use a little momentum and I rely a lot on the fundamental set ups of the market.

Do you have a mechanism that you screen for potential set ups and then you go in with discretion?

Not much discretion, it's pretty clear-cut. Like today, I had a specific buy at yesterday's high and when the price went to that point the signal said to go long, so I went long.

What kind of stop do you put on that?

Stops are interesting. What I've found to be the best type of stop, it's not some magical price point. I don't believe that. That's a bunch of bunk. I think the purpose of the stop is to protect your equity. My stops are based on dollar amounts. I know that I cannot lose more than x number of dollars, I don't care if there's a chart point real close or if that supposedly valid chart point is a mile away. It's not about my money, my money can only be protected absolutely, and it's much

more than a dollar stop than some mystical number. If I say I can only risk x number of dollars it's the fact that I can only lose x number of dollars.

Would you ever use something like the range of the market?

I would default to either the range or a volatility type of stop or x dollars. I can't go beyond x dollars. With a volatility stop you might have a stop that's so gargantuan that you get a huge loss, so you will default to x dollars.

What data format are you using with that?

I'm lying to my data so it looks like its twenty-eight years ago. At some point I'll get either TradeStation or I hear there is some other software coming out. I'll have to adjust. There's a lot to system writer, it does so many wonderful things that other software doesn't do.

Have you ever had any situations where you felt like being able to move the market was an advantage?

No. In the old days when I started trading stocks, the really, really old days in the sixties, the early sixties, if I would put a recommendation on the hotline, it could move the market, but there's no advantage to it because it becomes a blip in price.

You have to support it?

Right, it's an aberration and somebody's going to come in on the other side and say I'm selling. No advantage in that!

Imagine someone came up to you and said, "I want to become a money manager." What kind of advice would you give them?

I'd tell them to think long and hard about it because you have to have so many regulations. You have to have a personality for it. I don't have a personality that justifies existence with the United States government and regulatory bodies, accountants and lawyers. That's just not me. I'm not interested in that, but somebody that is and has that business background, which I don't have, it may be a good business but you better have a system, you better have a good modality to trade from. The first thing is, do you have the wherewithal to deal with bureaucrats?

So, the people that want to know about their money are a factor too?

Well, and you have to be a salesman, you have to be able to go out there and shuck and jive people and you have to keep their money and you have to work with brokerage firms. To me it's simpler and cleaner just to trade. I'll never make as much money as John Henry or Paul Tudor Jones. These guys have

amassed major fortunes that will last for generations. They have those skills. I don't have those skills. I barely have enough skills to trade, let alone run a major business.

So, you basically stick to bonds and S&P's, you're not thinking of switching to ten year's?

Doubt it, that's just for my own trading remember, when I do seminars or hotlines I'll talk about other markets and a longer-term approach than my own trading. The Ten Year looks like it's starting to pick up some volume. So you're going to have to take a look at it and see what's going on.

Do you think the arbitrage is bringing prices to where they are in check?

I just think my big deal is that you have to be where all of the speculative wave is. In the mid-eighties that wave left T-bills and went to Treasury Bonds. T-bills are what we traded back then. You have to watch the volume and see where everybody is. Where the party is, that's where I'm going.

What about the gold rush of the Nasdaq and everybody jumping on the bandwagon trading stocks? In your opinion is it easier to trade the S&P's?

The NASDAQ has been so wild! I would prefer to trade the S&P. It's probably a little tamer, but it's still wild, it's not a tame market by any stretch of the imagination but it's not as wild as the Nasdaq.

Are you developing any stock trading systems?

We have stock trading research. The latest thing I've been working on is a sentiment index for all stocks that have options on them. It's really cool. We've gone in to web pages and polled advisors to see what the consensus is. It's like market mayhem for commodities. We have that for stocks.

Are you going to do that on your ctiming.com site?

I don't know what's going to happen with it. I think Genesis Data will probably distribute it or Bloomberg.

Is that similar to a strategy using options where you want to own a stock at a certain price?

No, it's just an indicator. If a vast majority of advisors are either bullish or bearish we take the other side. These guys have a great record of being wrong.

So that's really your selection process?

Yes, and it's a great one.

Once you establish the selection process, then you go in with a momentum indicator or something similar?

Exactly!

Do you use the same money management techniques for a stock system as you do for futures?

You have to look at your equity and what your potential risk is and how much damage can be done. You know if you're going to get into a fight, you better see how much the guy can hurt you.

What do you tell a guy that comes up to you at the seminar and says how do I get started doing this?

You better read a bunch of books, start to paper trade first, they never want to paper trade, they think it's not the same as real trading and that's true. You know what? When guys go to medical school, I love to ask this question to doctor's in my audience, how many of you started to do surgery your first day of medical school? They have cadavers until their second or third year. We need some cadavers here to operate on and then see if this is for you. But to just rush in because you're smart and you made a bunch of money in your business or you got it from your parents or you inherited it, that's no reason to trade. You better learn the basics of this business before you start risking real money and that's the cool thing, you can do that! You can actually start to paper trade and see if this stuff fits you. Most people want to rush on in. This is the fast lane, fast track stuff you can get hurt doing this big time. So, I'd say go-slow.

Do you use the same approach to explain to somebody that this is just a day-to-day business? I talked to Miles and he said you guys are in there at five o'clock in the morning and we're on east coast time but it's still eight o'clock and you know, it's everyday. Do you try to stress that to people also?

Well, yeah, I think there's a lot of swash buckler's that have walked into this business and they think you can sit and stare at a quote machine and make money and it's work and you have to be alert and you have to come in, this is not nirvana, this is not an easy path to instant wealth. It sure beats the heck out of having a job or having a boss! It still has its up days and its down days. I've never yet found a way to make money where it's just real easy, where you never have to work or sweat about it, never have found that in my life, you're not going to find that I don't think and commodity trading is no different.

How much longer are you going to do it?

I enjoy this so much. I'll probably be trading for my entire life. To what point I remain a public figure is undetermined. I've started to back out of that a little bit. Miles Dunbar is taking over my newsletters. He's doing more and more all of the time and I'm doing less and less. At least that's the goal. At some point I need to fade into the woodwork. There are a lot of bright guys like Miles coming up that will blow me away. You know it's sad to see an athlete performing past his prime and this is no different.

Who do you see as the guys stepping up to fill in your shoes? It doesn't seem like there's anyone like John Hill and yourself. You guys are huge in the industry.

That's an interesting point. It could be because guys like John and I literally traipsed around this country lecturing and talking about commodities for so long, it's so ingrained in us. The new breed is highly computer, electronic, and stock oriented. They may not have the depth that guys like John have, of knowing what a soybean is, how gold is mined, and that may make a difference. Everybody has become so electronic, we old timers may have a little better understanding of things, you look at the money managers and there is some new blood coming up, it intrigues me how many of those people if you look at what their doing, they're doing things that John Hill was doing twenty, thirty years ago. They have a little variation and a little different insight, but they are not too far from where John and I have been all of these years.

So, really there's one way to do it, and if you can follow the path and stick with it you will be successful. Do you think adjusting your psychological makeup to be able to handle losses and let the winners run is the crux?

I think a lot of it is establishing your parameters. If you know going into trade how bad the loss can be, you can play the game. If you try and play football and it's not one hundred yards to the touchdown, sometimes it's ninety-five and sometimes it's a hundred and ten and sometimes there are boundaries and sometimes there aren't and sometimes there are penalties and sometimes there aren't. You will create insanity because people don't have a clue between what is right and wrong. Then they think, I know what, it's me, I need to go see a shrink, so, somebody does a psychobabble number on them. The reality is that if you have a relatively decent approach and you realize that the market's not perfect and you are never going to be right all of the time and you just accept that and you have basic rules, then I think you might be a trader. People have these unreal expectations that you can call the market as W.D. Gann supposedly did, though he didn't. We have evidence of that all over the place.

If that is your expectation and since you can never meet that expectation you'll go berserk as a trader. But if you just realize, hey this is a business of ups and downs, it's just like somebody that buys sweaters in the store and some of the sweaters sell and some of them don't sell. You have to take care of those losses, this is a business.

APPENDIX

A

EasyLanguage Syntax Errors

Syntax errors are produced when verifying an EasyLanguage statement that is not understood or expected by the PowerEditor. Following is a list of all syntax errors and their description, listed by error number. Each entry includes the description of the error, probable causes of the error, and examples of the correct and incorrect syntax for the offending statement or instruction (where applicable).

61 "Word not recognized by EasyLanguage."

This error is displayed whenever a word is not recognized by the PowerEditor. For example, if it is not an EasyLanguage reserved word; EasyLanguage function, or a declared user defined variable, array, or input name.

62 "Invalid number."

The PowerEditor displays this message whenever it finds a typographical error in a number. For example, if a letter is inserted by mistake in a number, the number will be highlighted and this error will be displayed. An example of an invalid number is *100.b4*.

63 "Number out of range."

The PowerEditor displays this error whenever it finds a number that is outside the supported range (a number which is too big). The following statement will produce this error:

```
Value1 = 99999999999999999999;
```

65 "Invalid variable name."

The PowerEditor displays this error whenever it finds an invalid name in a variable declaration statement. Variable names cannot start with a number nor any special character other than the underline (_).

For example, this error will be generated when the following statement is verified:

```
Variable: $MyVariable(0);
```

66 "Invalid input name."

The PowerEditor will display this error whenever it finds an invalid name in an input declaration statement. Input names cannot start with a number nor any special character other than the underline (_).

For example, this error will be generated when the following statement is verified:

```
Input: $MyInput(0);
```

70 "Array size cannot exceed 2 billion elements."

Arrays can have up to 2 billion elements. The number of elements is calculated by multiplying all the dimensions of the array. For example, an array declared using the following statement will have 66 elements:

```
Array: MyArray[10,5](0);
```

This arrays will have rows 0 through 10 and columns 0 though 5; in other words, 11 rows and 6 columns. The resulting number from multiplying the dimensions of the array can't exceed 2 billion.

74 "Invalid array name."

The PowerEditor displays this error whenever it finds an invalid name in an array declaration statement. Array names cannot start with a number nor any special character other than the underline (_).

For example, this error will be generated when the following statement is verified:

```
Array: $MyArray[10](0);
```

90 "The first jump command must be a begin: (\\hb,\\pb,\\wb)"

This error is displayed when the PowerEditor finds an end *jump command* without a begin *jump command* in a text string. The end jump commands are:

```
\he
\pe
\we
```

Before these commands, a begin *jump command* must be used.

***Note**: when specifying a file name for the Print() or FileAppend() words, files that start with any of the jump commands will produce this error. So a file name "c:\hello.txt" will produce this error as part of the name \he.*

91 "You cannot nest jump commands within other jump commands."

Jump commands are used in commentary-related text string expressions to highlight words, and create links to the on line help. *Jump commands* cannot be nested; that is, there cannot be multiple starting *jump commands* without having matching end *jump commands*.

92 "You must terminate all jump commands with ends (\\he,\\pe,\\we)"

This error is displayed when the PowerEditor finds a begin *jump command* without an end *jump command* in a text string. The begin *jump commands* are:

```
\hb
\pb
\wb
```

After these commands, an end *jump command* must be used.

***Note**: when specifying a file name for the Print() or FileAppend() words, files that start with any of the jump commands will produce this error. So a file name "c:\hello.txt" will produce this error as part of the name \he.*

151 "This word has already been defined."

User defined words (such as variables, arrays, and inputs) need to have unique names. This error is generated when a user defined word is defined more than once, such as in the following example:

```
Input: vac(10);

Variable: vac(0);
```

154 "=, <>, >, >=, <, <= expected here."

This error is displayed when the PowerEditor evaluates complex true/false expressions and it finds an error within the expression.

```
Condition1 = Condition2 = Close;
```

The intention of this statement was to assign a complex true-false value to the variable *Condition1*, by using *Condition2* and a comparison that involves the *Close*. A corrected version would look like this:

```
Condition1 = Condition2 AND Open = Close;
```

155 " '(' expected here."

The left parenthesis was expected before the highlighted word; for example, if you are using a function that requires parameters, and no parameters are listed.

```
Value1 = Average + 10;
```

In this example, the highlight signifies that a parenthesis was expected before the '+' sign.

156 " ')' expected here"

The right parenthesis was expected after the highlighted word; for example, if you are using a function that requires parameters, you must enclose them in parentheses.

```
Value1 = Average(Close, 10;
```

Here, the highlight signifies that a closing parenthesis was expected before the';'

157 "Arithmetic (numeric) expression expected here."

This error is displayed whenever the PowerEditor is expecting a number or a numeric expression and it finds a true-false expression, string value, or any other keyword that does not return a numeric expression. For example, the *Average()* function expects two numeric expressions, so the following:

```
Value1 = Average(Condition1, 10);
```

generates an error since *Condition1* is a true-false expression.

158 "An equal sign '=' expected here."

This error is displayed if the equal sign is omitted when assigning a value to a variable, array, or function (writing an assignment statement).

For example, the following statement will cause an error:

```
Value1 10;
```

and would be corrected by adding an equal sign, as in:

```
Value1 = 10;
```

159 "This word cannot start a statement."

Not all words can be used to start a statement. For example, the data word *Close* cannot be used to start a statement. Usually, reserved words that generate some action are used to start statements such as *Buy*, *Plot1*, or *If-Then*.

160 "Semicolon (;) expected here."

All EasyLanguage statements must end with a semicolon. Whenever the PowerEditor finds a word or expression that can be interpreted as a new line, it will place the cursor before this expression and show this error. For example, the following statements will produce this error:

```
Value1 = Close + Open
Buy Next Bar at Value1 Stop;
```

Given that the word *Buy* is always used at the beginning of a statement to place a trading order, a semicolon is required after the *Open*.

161 "The word THEN must follow an If condition."

This error is displayed whenever the word *Then* is omitted from a *If-Then* statement. The word *Then* must always follow the condition of the *If-Then* statement. The correct syntax for an If-Then statement is:

```
If Condition1 Then {any operation}
```

162 "STOP, LIMIT, CONTRACTS, SHARES expected here."

This error is displayed by the PowerEditor if it finds a numeric expression following a trading verb without including one of the words listed above. A numeric expression can be used in a trading order to determine the number of shares (or contracts) and/or to specify the price of the stop or limit order. For example:

```
Buy Next Bar at Low - Range;
```

is incorrect because it does not include a trading verb after the price **Range**. To be correct, you could add the word Stop or Limit, as in:

```
Buy Next Bar at Low - Range Stop;
```

163 "The word TO or DOWNTO was expected here."

This error is displayed whenever writing a *For* loop and the word *to* or *downto* is omitted. The correct syntax for a *For* loop is:

```
For Value1 = 1 To 10 Begin
   {statements}
End;
```

165 "The word BAR or BARS expected here."

This error is displayed whenever referencing to a value of a previous bar where the word *Bar* is omitted. For example, the following statement will cause this error:

```
Value1 = Close of 10 Ago;
```

The correct syntax is:

```
Value1 = Close of 10 Bars Ago;
```

166 "The word AGO expected here."

This error is displayed when the PowerEditor finds a reference to any expression for a number of bars ago without using the phrase *Bars Ago*. For example:

```
Value1 = Close of 10 Bars;
```

produces this error because the word *Ago* is missing. The correct syntax for this expression is:

```
Value1 = Close of 10 Bars Ago;
```

167 " '}' was expected before end of file."

In order to add comments to your EasyLanguage, it is necessary to enclose the commentary text in the curly braces '{' and '}'. An error message is displayed when a left curly brace is found without a matching right curly brace.

```
{ this was written by Trader Joe
If Close > Highest(High, 10)[1] Then
   Buy Next Bar at Market;
```

Above, the right curly brace was omitted somewhere before the vertical cursor. In this example a right curly brace should have been placed after the word 'Joe'.

168 " '[' was expected here."

When declaring, assigning, or referencing array values you are required to use the squared braces to specify the array element(s). This error is displayed if the left squared brace is not used when working with an array.

```
Array: MyArray(10);
```

For example, here the highlight shows that a squared brace, corresponding to the declared number of array element, is expected before the parenthesis.

169 "']' was expected here."

When working with bar offsets or arrays, the bar or array index must be enclosed in squared braces. This message is displayed if the right squared brace is missing.

```
Value1 = Close[10 * 1.05;
```

In this example, the highlight indicates that a squared bracket should be placed somewhere before the semicolon. Note that since the PowerEditor is expecting a numeric value in the squared braces, it places the highlight after the last character in a numeric expression. However, in this case, the right bracket was probably intended to be placed after the number 10.

170 "Assignment to a function not allowed."

This error is displayed when you attempt to assign a value to a function. By definition, a function is an EasyLanguage procedure that returns a value, so it is not possible to assign a different value to a function (except when returning a value from within a function).

```
Average = 100.1245;
```

In this example, the highlighted function name indicates that you cannot assign it a value.

171 "A value was never assigned to user function."

By definition, a function is a set of statements that return a value. This error will be displayed when editing or creating a function and the PowerEditor finds that no value has been assigned to the function. A statement similar to the following must be included in every function:

```
MyFunction = Value;
```

where *MyFunction* is the name of the function and *Value* is the expression to be returned when the function is referenced.

172 "Either NUMERIC, TRUEFALSE, STRING, NUMERICSIMPLE, NUMERICSERIES, TRUEFALSESIMPLE, TRUEFALSESERIES, STRINGSIMPLE, or STRINGSERIES expected."

When declaring the inputs in a function it is necessary to specify the type of each input. This error is generated when any word or value, other than a valid input type, is used when declaring function inputs.

174 *"Function not verified."*

In order for an analysis technique to verify, all functions used by the analysis technique must be verified as well. This error is displayed if there is a function that is not verified and you attempt to verify the analysis technique.

In order to solve this, open the function and verify it, or run "Verify All" from the PowerEditor menu.

175 " ',' or ')' expected here."

This error is displayed when listing a number of elements in parentheses and a semicolon is read before the list is finished.

```
Value1 = Average(Close, 10;
```

In this case, the highlight indicates that either more parameters (separated by a comma) or a right parenthesis were expected before the semicolon.

176 "More inputs expected here."

This error is displayed whenever referencing a function or an included strategy without specifying enough inputs. For example:

```
Value1 = Average(Close);
```

displays an error because only one input is specified while the *Average* function requires two inputs: 1) the price to be averaged and 2) the number of bars.

177 "Too many inputs supplied."

The PowerEditor displays this error when too many inputs are supplied for a function. For example, the *Average* function requires only two inputs, so the following statement will produce this error:

```
Value1 = Average(Close, 10, 5);
```

The correct syntax would be

```
Value1 = Average(Close, 10);
```

180 "The word #END was expected before end of file."

The compiler directive #END must be used to indicate the end of a group of statements included in the alert or commentary only section of an analysis technique. The alert and commentary compiler directives will allow certain instructions to be executed only when the alert or commentary is enabled.

181 "There can only be 10 dimensions in an array."

Arrays can have up to 10 dimensions. The correct syntax for creating a multi-dimensional array is:

```
Array: MyArray[10,10,10](0);
```

where this statement creates a three dimensional array of 11x11x11

183 "More than 100 errors. Verify termination."

When the PowerEditor is verifying a document for correctness, it will continue to evaluate expressions until it finds 100 errors. These errors will be found in the error log once the verification process is finished. If the PowerEditor finds more than 100 errors it will stop the process and will display this message.

185 "Either HIGHER or LOWER expected here."

When specifying the execution instructions for an order in a strategy, it is possible to use the words *or Higher* and *or Lower* as synonyms to stop and limit. This error occurs when the word *or* is found in an order without the words *Higher* or *Lower*. The following is the proper syntax for this statement:

```
Buy Next Bar at Low - Range or Lower;
```

186 "Input name too long."

Input names in any PowerEditor analysis technique can be up to 20 characters long. This error is displayed by the PowerEditor whenever an input has a name that has more than 20 characters.

187 "Variable name too long."

Variable names can have up to twenty characters. This error is displayed whenever a variable is declared with a name that contains more than twenty characters.

188 "The word BEGIN expected here."

This error is generated whenever the PowerEditor is expecting a block statement. For example, all loops require *Begin* and *End* block statements, so writing the following will generate this error:

```
For Value1 = 1 To 10
    Value10 = Value10 + Volume[Value1];
```

The correct syntax is:

```
For Value1 = 1 To 10 Begin
   Value10 = Value10 + Volume[Value1];
End;
```

189 "This word not allowed in a strategy."

The word highlighted by the PowerEditor is not allowed in a Strategy.

190 "This word not allowed in a function."

The word highlighted by the PowerEditor is not allowed in a function. Words like *Plot1*, *Buy*, *SellShort*, etc., are not allowed in functions.

191 "This word not allowed in a study."

The word highlighted by the PowerEditor is not allowed in a study. Words like *Plot1*, *Buy, SellShort*, etc., are not allowed in studies.

192 "This word not allowed in an ActivityBar."

The word highlighted by the PowerEditor is not allowed in an ActivityBar study. Words like *Plot1*, *Buy*, *SellShort*, etc., are not allowed in ActivityBar studies.

193 "Comma (,) expected here."

Commas are used to separate elements in a list; for example when declaring multiple inputs or variables, or when listing the parameters of a function.

This error will be generated whenever the PowerEditor finds two words, that seem to be part of the list, which are not separated by a comma. For example, in the following:

```
Inputs: Price(Close) Length(10);
```

the comma after the first input is missing. The PowerEditor places the vertical cursor at the location where it was expecting a comma.

195 "Matching quote is missing."

All text string expressions need to be within double quotes. This error will be displayed whenever there are not matching quotes around a text string expression. For example, the following statement will produce this error:

```
Variable: Txt(" ");
Txt = "This is an example;
```

because there is a missing quote to the right of the text expression. The correct syntax for this expression is:

```
Variable: Txt(" ");
Txt = "This is an example";
```

197 "Strategy not verified."

In order for a trading strategy to verify, any strategies referenced by the trading strategy through the use of the *IncludeStrategy* reserved word must be verified as well. This error is displayed if you attempt to verify a trading strategy that references an unverified strategy.

In order to solve this, open the referenced strategy and verify it, or run "Verify All" from the PowerEditor menu.

200 "Error found in function."

This error is displayed whenever verifying an analysis technique that refers to an unverified function. The only solution is to open the function, verify the function, and then return to the analysis technique.

201 "User function cannot refer to current cell of itself."

A simple function cannot refer to the same value of a function within its calculations. However, if defined as a series function, it can refer to a previous value of itself. For example, the following simple function gives an error:

```
MyFunction = MyFunction + Volume;
```

because the calculation refers to the current value of the function. By setting the function **Parameter** to "Series", the following becomes a valid expression that uses a function's previous value to accumulate the volume of the chart:

```
MyFunction = MyFunction[1] + Volume;
```

204 "Orders cannot be inside a loop."

EasyLanguage does not allow trading orders to be placed inside a *For* or *While* loop. If the intention of placing an order inside a loop is to increase the number of shares or contracts that the strategy will handle, this can still be done by placing the calculation of the number of shares or contracts inside a loop and then using the resulting value in the order instruction after the loop is finished. For example,

```
While Condition1 Begin
   Value1 = <calculation of number of shares>;
End;
Buy Value1 Shares Next Bar at Market;
```

205 "Statement does not return a value."

This error is displayed when attempting to return a value from statements not designed to return a value, such as those that set or change a value. For example:

```
Value1 = AB_SetZone(High, Low, RightSide);
```

To correct this error, do not assign the expression to a variable:

```
AB_SetZone(High, Low, RightSide);
```

208 "CONTRACTS, SHARES expected here."

When writing an EasyLanguage statement to place an order, it is possible to specify how many contracts or shares the strategy should use to open (or exit) the position. This error will be generated by the PowerEditor whenever it finds a numeric expression after the trading verb that is not followed by the words *Stop, Limit, or Higher,* or *or Lower*. For example:

```
Buy 100;
```

generates an error because it is not clear if '100' is a part of the instructions to specify the number of shares or the execution instruction (the price at which the order should be placed). A correct statement might read:

```
Buy 100 Shares;
```

209 "Strategy name expected within quotes."

When specifying the name of an order, it must be enclosed within parentheses and double quotes. This error is displayed if the name is missing or not correctly provided. For example, the following statement will cause this error:

```
Sell From Entry () Next Bar at Market;
```

211 "Strategy cannot call itself."

A strategy cannot reference itself when using the *IncludeStrategy* reserved word.

213 "Error found in strategy."

This error is displayed whenever verifying a strategy that contains the *IncludeStrategy* reserved word which references a strategy that is not verified. The only solution is to open the unverified strategy, verify it, and then return to the original strategy.

214 "Colon (:) expected here."

EasyLanguage expects a colon to be used when declaring certain elements of the language like inputs, variables, arrays, and DLLs. In order to declare a new input, the word input should be followed by a colon, and then the list of input names. This error will be displayed whenever the colon is missing from this expression, for example:

```
Input MyValue(10);
```

Since there is no colon after the word 'Input', the word *MyValue* is highlighted and this error message is displayed. To correct the error, simply add a colon after 'Input':

```
Input: MyValue(10);
```

215 "Cannot use next bar's price and close order in the same strategy."

EasyLanguage does not support using information from the next bar (the *Date*, *Time*, or *Open*) and placing an order at the close of the current bar in the same strategy. If the instructions are not related, they should be written as different strategies and merged using TradeStation StrategyBuilder.

The following produces an error because it includes a reference to the *Open of Next Bar* with a *Close* order for *This Bar* (current bar):

```
If Open of Next Bar > Price Then Buy This Bar on Close;
```

217 "Function circular reference found."

A circular reference is defined as two formulas that refer to each other in their respective calculations. This type of formula cannot be solved by EasyLanguage, so whenever a circular reference is found this error is displayed.

For example, a circular reference can happen if you have a function *A,* which is defined as the value of the current bar of a function *B* plus 1, and the definition of the function *B* is the value of the current bar of *A* plus 1. In order to calculate the value of function *A*, the value of *B* is needed, but in order to calculate *B*, the value of *A* is needed. Therefore, it is not possible to obtain the values of these functions and this error occurs.

220 "Cannot anchor a global exit."

The price date of the bar where an entry order was placed can be accessed from an exit by using *At$*. This is only allowed when the entry order has a label and if the exit is tied to the entry. An error will be generated if the entry is not labeled or if the exit does not specify what entry it is attempting to close. For example, the following exit will cause this error:

```
If Condition1 Then
   Buy ("MyEntry") This Bar on Close;
Sell At$ Low - 1 Stop;
```

since the exit does not specify the name of a matching entry. The correct syntax is:

```
If Condition1 Then
   Buy ("MyEntry") This Bar on Close;
Sell From Entry ("MyEntry") At$ Low - 1 Stop;
```

223 "A simple function cannot call itself."

Historical values of simple functions are not available to EasyLanguage, so referring to previous values of itself in its calculations is not allowed. If this is necessary, change the function to a series.

```
MyFunction = MyFunction[1] + Volume;
```

For example, if *MyFunction* is a simple function, the above reference to the value of *MyFunction* of one bar ago is not allowed in this calculation.

224 "Strategy name already used."

The PowerEditor does not allow the reuse of a name in two different orders. It is mandatory that all orders have a different name. The following *SellShort* statement produces this error:

```
If Condition1 Then
   Buy ("MyStrategy") Next Bar at Market;

If Condition2 Then
   SellShort ("MyStrategy") Next Bar at Market;
```

because both orders cannot have the same name.

226 "Next bar's prices can only be used in a strategy."

The *Open, Date* and *Time* of the next bar can only be referenced from a strategy; no other analysis has access to this information.

227 "Default expected here."

When declaring an input in any analysis technique, you need to enclose the default value in parentheses. This error will be shown whenever there is no default value specified (the parentheses are empty). For example, the following is the correct syntax for declaring an input with the default value of 15:

```
Input: MyInput(15);
```

229 "Invalid initial value."

An initial value needs to be specified when declaring a variable or array. This initial value needs to be enclosed by parentheses and is used to 1) determine the type of the variable or array (numeric, true-false, or text string), and 2) assign the initial value of the variable or array on the first bar.

The correct syntax when declaring a variable is:

```
Variable: MyVariable(10);
```

where the initial value assigned to this variable is *10,* which is a numeric value.

230 "Initial value expected here."

An initial value needs to be specified when declaring a variable or array. This initial value needs to be enclosed by parentheses and is used to 1) determine the type of the variable or array (numeric, true-false, or text string), and 2) assign the initial value of the variable or array on the first bar.

The correct syntax when declaring a variable is:

```
Variable: MyVariable(10);
```

where the initial value assigned to this variable is *10,* which is a numeric value.

231 "Function has no inputs. Parenthesis not needed."

This error is shown by the PowerEditor when parentheses are used for a function which has no inputs. For example, the EasyLanguage function *Range* has no inputs, so the following statement:

```
Value1 = Range(10);
```

displays the error message and highlights the first parenthesis before the parameter.

232 "Matching left comment brace '{' is missing."

The PowerEditor displays this error whenever it finds a right comment brace "}" without a matching left comment brace. In order to fix this, find the beginning of the comment text and place a left comment brace before it. If there is no comment in your analysis technique, then remove the right comment brace.

233 "Extra right parenthesis."

When writing any type of expression or statement that requires parentheses, it is necessary to have matching left and right parentheses. This error is displayed if there are extra right parentheses in the expression being evaluated. For example:

```
Value1 = (Close + Open))/2
```

234 "END found without matching BEGIN."

This error is displayed whenever a block statement does not contain a matching *End* for every *Begin*.

237 "Position Information function not allowed in a study."

Strategy position information words can only be used in strategies and functions. This error will be generated if any one of these words are found in anything other than a strategy or function.

238 "Performance Information function not allowed in a study."

Strategy performance information words can only be used in strategies and functions. This error will be generated if any one of these words are found in anything other than a strategy or function.

239 "Array name too long."

Array names can have up to 20 characters. An error message will be displayed if the array name used in the declaration statement has more than 20 characters.

240 "This strategy name does not exist."

This error is displayed whenever tying an exit to a non-existent entry name. For example, the following strategy produces this error:

```
Buy ("Break") Next Bar at Highest(High, 10) Stop;

Sell From Entry ("BreakOut") Next Bar at Low Stop;
```

because the exit incorrectly refers to an entry labeled "BreakOut" which does not exist in this strategy. Changing the entry name to "Break" will correct this error.

241 "Cannot exit from an exit strategy."

This error is displayed when an exit strategy is mistakenly tied to another exit strategy. Exit strategies can only be tied to an entry through the use of the instruction *from Entry ("entry name")*. For example, the following statements will generate this error:

```
If Condition1 Then
   Buy ("MyEntry") This Bar at Close;

If Condition2 Then
   Sell ("MyExit") This Bar at Close;

Sell from Entry ("MyExit") Next Bar at Lowest(Low,10) Stop;
```

Instead, the following statements are correct:

```
If Condition1 Then
   Buy ("MyEntry") This Bar at Close;

If Condition2 Then
   Sell ("MyExit") This Bar at Close;

Sell From Entry ("MyEntry") Next Bar at Lowest(Low,10) Stop;
```

242 "Cannot BuyToCover from a buy strategy."

This error will be displayed when a short exit strategy is tied mistakenly to a long entry strategy. Short exit strategies can be tied only to a short entry through the use of the instruction *from Entry ("entry name")*. For example, the following statements will generate this error:

```
If Condition1 Then
   Buy ("MyEntry") This Bar at Close;
BuyToCover From Entry ("MyEntry") Next Bar at Lowest(Low,10)
Stop;
```

In this case, the error can be corrected by using the appropriate exit instruction, *Sell*:

```
If Condition1 Then
   Buy ("MyEntry") This Bar at Close;
Sell From Entry ("MyEntry") Next Bar at Lowest(Low,10) Stop;
```

243 "Cannot Sell from a SellShort strategy."

This error will be displayed when a long exit strategy is tied mistakenly to a short entry strategy. Long exit strategies can be tied only to a long entry through the use of the instruction *from Entry ("entry name").* For example, the following statements will generate this error:

```
If Condition1 Then
   SellShort ("MyEntry") This Bar at Close;

Sell from Entry ("MyEntry") Next Bar at Low Stop;
```

In this case, the error can be corrected by using the appropriate exit instruction, *BuyToCover*:

```
If Condition1 Then
   SellShort ("MyEntry") This Bar at Close;

BuyToCover from Entry ("MyEntry") Next Bar at Low Stop;
```

244 "At$ cannot be used after the word TOTAL."

EasyLanguage does not allow the reserved word *Total* to be tied to reference information from the bar of entry by using the *AT$* instruction. For example, the following statement will generate this error:

```
Sell 20 Shares Total From Entry ("MyEntry") At$ Low Stop;
```

247 "References to previous values are not allowed in simple functions."

Prior values of simple functions, simple variables, or simple expressions cannot be referenced from within simple functions. If this is necessary for the calculation of a function then the function must be set as series, not simple. This incorrect example:

```
MyFunction = MyFunction[1] + Close;
```

creates an error if *MyFunction* is a simple function with a reference to previous values of itself. Setting the function **Properties** to "Series" will correct this error.

250 "Cannot reference a previous value of a simple input."

Historical values of simple inputs in functions are not stored by EasyLanguage, so referring to previous values of them is not allowed. For example, in the following:

```
Input: MyVal(NumericSimple);
MyFunction = MyVal[5];
```

the value *MyVal[5]* is not allowed in this function since it includes a reference to the value of the input of five bars ago but is declared as a *NumericSimple* input. If the reference to a previous value is necessary, change the input type to series.

251 "Variables and arrays not allowed here."

In previous product versions this error is displayed when attempting to pass variables or arrays to series functions.

```
Value2 = Average(Close, Value1);
```

253 "Cannot reference a previous value of this input."

Historical values of simple inputs in functions are not stored by EasyLanguage, so referring to previous values of them is not allowed. For example, in the following:

```
Input: MyVal(NumericSimple);
MyFunction = MyVal[5];
```

the value *MyVal[5]* is not allowed in this function since it includes a reference to the value of the input of five bars ago but was declared as a *NumericSimple* input. If the reference to a previous value is necessary, change the input type to series.

258 "Variables, arrays and inputs not allowed here."

This error is displayed when a variable, array, or input is used as the initializer for an input value, such as:

```
Vars: MyVar(3);
Input: MyInput(MyVar);
```

259 "This number is too big."

The PowerEditor displays this error whenever it finds a number that is outside the supported range (a number which is too big). The following statement produces this error:

```
Value1 = 99999999999999999999;
```

260 " 'Next Bar' can only be applied to 'OPEN', 'DATE' and 'TIME'."

The only prices available from the next bar that can be read from a strategy are *Open, Date,* and *Time.* All other prices from the next bar can't be accessed.

261 "The word 'BAR' expected here."

This error is shown whenever writing an order in a strategy where the word *Bar* is left out of the expression. For example, the following:

```
Buy Next on the Close;
```

generates an error because *Bar* is missing. The correct syntax is:

```
Buy Next Bar on the Close;
```

262 "At market order can only be placed for the next bar."

All analysis techniques are read and executed at the end of each bar. Because of this, market orders can only be placed for the next bar. An error will be generated whenever a market order is placed to be filled on this bar, such as:

```
Buy This Bar at Market;
```

263 "Stop and limit orders can only be placed for the next bar."

This error is displayed when trying to write a stop or limit order for the current bar. For example:

```
Buy This Bar at Low - Range Limit;
```

is not correct because a *Limit* order cannot be placed on *This Bar*. To be correct, the *Limit* order must be on the *Next Bar*:

```
Buy Next Bar at Low - Range Limit;
```

264 "On close order must be placed for this bar."

Given that all instructions are read at the close of each bar, the only types of orders that can be placed on the current bar are at the close. Whenever *This Bar* is included as part of an order it may only refer to the *at Close* price. The correct syntax for *This Bar* orders is:

```
Buy This Bar at Close;
```

265 "Cannot mix next bar prices with data streams other than data1."

EasyLanguage prohibits the reference of secondary data streams in the same strategy where references to the *Date*, *Time*, or *Open* of the next price are also made. If the references to a secondary data stream and the next bar prices are not directly related, it is recommended that you write two strategies, one that uses next bar prices and a second that references other data streams.

For example, the following statements included in one strategy are not allowed because they reference two different data streams (*Data1* by default is the first and *Data2* in the second):

```
If Open Next Bar > High Then
   SellShort Next Bar at Open Next Bar + Range Limit;

If Average(Close, 4) of Data2 < Average(Close, 7) of Data2 Then
   BuyToCover Next Bar at Close;
```

Instead, writing two different strategies, one containing the first IF-THEN statement and another containing the second IF-THEN statement is necessary. Later these strategies can both be included as part of the same Trading Strategy by adding them to the same Chart Analysis window.

266 "Library name within double quotes expected here."

The PowerEditor displays this error when defining an external DLL function and the name of the DLL is missing or incorrect. The first element of the list of parameters in the *DefineDLLFunc* statement should be the name of the DLL library within double quotes. The following statement will generate this error:

```
DefineDLLFunc: int, "MyFunc", int;
```

The correct syntax for this statement is:

```
DefineDLLFunc: "MyDLL", int, "MyFunc", int;
```

267 "DLL function name within double quotes expected here."

When defining a function from a DLL, the name of the DLL must be enclosed in double quotes. For example, the following is a proper example of such a function definition because it includes the function name "user.dll" followed by DLL's return type and parameters:

```
DefineDLLFunc: "user.dll", int, "beep";
```

274 "Return type of this DLL function must be specified."

When declaring a DLL function, the return type of the function must be the second parameter listed. Following is a correct DLL function declaration statement with the DLL's type **int** following the DLL name:

```
DefineDLLFunc: "MyDLL.DLL", int, "MyFunction", int;
```

276 "DLL name cannot be longer than 60 characters."

The name of the DLL used to define any function through the *DefineDLLFunc* statement cannot exceed 60 characters**.**

277 "DLL function name cannot be longer than 65 characters."

The name of a function defined using the *DefineDLLFunc* statement cannot exceed 65 characters.

278 "A variable expected here."

Whenever the PowerEditor expects a variable and finds another reserved or user defined word, it will highlight the unexpected word and give this message. An example is when a function is expecting a variable as one of the parameters (because it is expecting to receive the variable by reference).

279 "An array expected here."

Functions can now receive arrays as parameters. If a function is expecting an array and instead the PowerEditor finds a variable, input, or other reserved word (different than an array), it will display this error. In the following example the function *Average_a()* calculates the average of a particular array, so the following will generate the syntax error:

```
Variable: MyVar(0);
Value1 = Average_a(MyVar, 10);
```

To correct this problem, you need declare *MyVar* as an array instead of an integer. It should be written:

```
Array: MyArray[20](0);
Value1 = Average_a(MyArray, 10);
```

280 "TrueFalse expression expected here."

This error is displayed when the PowerEditor expects a true/false expression and finds a numeric or text string expression instead. For example:

```
Condition1 = High ;
```

281 "Mixing data types (NUMERIC, TrueFalse, String) not allowed."

This error appears when incompatible data types are combined in a single expression.

In this example:

```
Value1 = 100 + "12" ;
```

the text string "12" cannot be directly combined with a numeric value. To resolve such a problem, use the appropriate EasyLanguage reserved word to convert the data to a compatible type.

For example, use the function *StrToNum* to convert the text string to a numeric value:

```
Value1 = 100 + StrToNum("12") ;
```

283 "Strategy has no inputs. Comma not needed."

When including a strategy through the *IncludeStrategy* keyword, the list of the inputs must be supplied and each input must be separated by a comma. This error is displayed if the strategy has no inputs, and an input is mistakenly included in the statement.

Following is the correct syntax of an *IncludeStrategy* statement of a strategy with no inputs:

```
IncludeStrategy: "My Trailing LX";
```

284 "There is no such strategy."

This error is displayed by the PowerEditor whenever the strategy name referenced by an *IncludeStrategy* statement does not exist in the strategy library.

285 "Strategy circular reference found."

A circular reference is defined as two formulas that refer to each other's current bar value in their respective calculations. This type of formula cannot be solved by EasyLanguage, so whenever a circular reference is found this error is displayed.

286 "Cannot divide by zero."

This error will be displayed when dividing any numerical expression by the literal number 0. So when the following is written:

```
Value1 = Close / 0;
```

the PowerEditor will generate a syntax error because dividing by zero is a mathematical indetermination and cannot be solved.

287 "File name expected here."

This error is displayed when using the *Print* statement to send information to the printer, and an invalid file name is used for the file name. The file name should be specified as text between double quotes. Note that a text string expression will not be accepted as a file name in the *Print* statement. For example, the PowerEditor will display this error when evaluating the following statement:

```
Print(File(Value1), Date, Time, Close);
```

The file name needs to be text included in double quotes; for example:

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

288 "A file or directory name must be <260 characters and may not contain "/ : * ? < > |"."

Certain instructions like the *Print()* and *FileAppend()* statements require a file name. The file name needs to be less than 260 characters long and cannot have any of the characters listed in the error label. For example, this error will be displayed when writing:

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

since the '?' character is not a valid character and cannot be used as part of a file name.

291 "The word 'OVER' or 'UNDER' expected here."

This error is displayed whenever using the word *Cross* without *Over* or *Under* when writing a true-false expression. For example, the following expression will produce this error:

```
Condition1 = Close Crosses Open;
```

The correct syntax would be:

```
Condition1 = Close Crosses Over Open;
```

292 "Two constants cannot cross over each other."

The PowerEditor displays this error whenever using the logical operators *Crosses Over* or *Crosses Under* compares two constants. Since they are constants, they will never cross each other and the statement will display an error, as in:

```
Condition1 = 10 Crosses Over 15;
```

293 "This plot has been defined using a different name."

The value of a plot can be assigned more than once within an analysis technique but it must always be referenced using the same name (or the name can be left out). For example, the following statement will cause this error:

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

If Volume > 1000000 Then
   Plot1(Volume, "V", Red);
```

because the plot has been assigned a second name "V". The correct way of writing this statement is:

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

If Volume > 1000000 Then
   Plot1(Volume, "Vol", Red);
```

295 "This plot name has never been defined."

This error is displayed whenever referencing a *Plot* with a different name than it was defined with, or a plot that doesn't exist. For example, the following statements will cause this error:

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

Value1 = Plot1 + Plot2 ;
```

since Plot2 has not been defined. The PowerEditor highlights the second instance of the *Plot* command to indicate where the error occurred.

296 "This plot has never been assigned a value."

This error is generated when referring to the value of a plot that has not been previously defined in the analysis technique. For example, the following statements will produce this error:

```
Plot1( Average(Close,10) );
If Plot1 Crosses Over Plot2 Then
   Alert;
```

because Plot2 has not been defined.

297 "Server field name too long; cannot be more than 30 characters."

Server Quote fields can be up to 30 characters long. This error will be generated whenever a server field with a name that has more than 30 characters is used.

298 "Strategy Information (for plots) function not allowed in a strategy."

None of the "Strategy Information for plots" words can be used within a strategy. These words are designed to be used in other analysis techniques to refer to overall performance of the strategy. However, there are strategy-specific words that can be used from the strategy to refer to these figures.

These words are:

```
I_AvgEntryPrice

I_ClosedEquity

I_CurrentContracts

I_MarketPosition

I_OpenEquity
```

299 "Strategy Information function not allowed in a study."

Strategy information words (other than the strategy information for plots) can only be used in trading strategies and functions. These words, which are listed in the EasyLanguage Dictionary under the categories *Strategy Performance* and *Strategy Position*, can only be used when writing trading strategies and functions.

300 "This plot has been defined with a different type."

The value of a plot can be assigned more than once but it must always be of the same type. Plot statements can display numeric, true-false, and string expressions, but they cannot change types within an analysis technique. For example, the following pair of *Plot* statements are not allowed in an analysis technique because they include different data types, where the first plot is a text string and the second a true-false value:

```
Plot1( "This is a text string");

If Condition1 Then
   Plot1(Condition1);
```

302 "Different number of dimensions specified in the array than the parameter."

This error is shown when an array is passed into a function with the wrong number of dimensions. For example, this error will be generated if a function is expecting a single dimension array but is sent an array with two dimensions instead.

303 "Extraneous text is not allowed after the array-type parameter"

When passing an array into a function, only the array name should be used. This error is displayed whenever any text, words, or braces are added after the array name that is passed to a function. For example:

```
Array: MyArray[10](0);

Value1 = Average_a(MyArray[0], 10);
```

the *[* will be highlighted because an array index appears after the array name. The correct syntax would be:

```
Array: MyArray[10](0);
Value1 = Average_a(MyArray, 10);
```

304 "Numeric-Array Parameter expected here."

Functions can receive arrays as parameters. If a function is expecting an array, any other type of parameter (variable, input, or reserved word) will display this error. In the following example:

```
Variable: MyVar(0);

Value1 = Average_a(MyVar, 10);
```

the function *Average_a()* requires an array on which to calculate an average and displays this error because *MyVar* is not an array.

Instead, you can write:

```
Array: MyArray[20](0);
Value1 = Average_a(MyArray, 10);
```

305 "TrueFalse-Array Parameter expected here."

Functions can now receive arrays as parameters. If a function is expecting a true-false array and, instead, the PowerEditor finds a variable, input, or other reserved word (different than a true-false array), it will display this error. For example, a function *MyTrueFalse_a()* that correctly uses true-false arrays would be written as follows:

```
Array: MyArray[20](False);
Variable: MyTF(False);

MyTF = MyTruefalse_a(MyArray, 10);
```

306 "String Array Parameter expected here."

Functions can now receive arrays as parameters. If a function is expecting an array of text strings and, instead, the PowerEditor finds a variable, input, or other reserved word (different than an array of text strings) it will display this error. For example, a function *Average_a(),* which combines all the text strings that are in an array into one, should be used as follows:

```
Array: MyArray[20](" ");
Variable: MyText(" ");
MyText = Average_a(MyArray, 10);
```

307 "The word 'Cancel' must be followed by 'Alert'."

Whenever canceling a previously enabled alert, the statement *Cancel Alert* needs to be used. This error is displayed whenever using the word *Cancel* without the word *Alert.*

314 "This word is only allowed in ActivityBar studies."

The words that are used to set the properties and draw ActivityBars are only allowed from ActivityBars and are not allowed in any other study or strategy.

323 "'Value-type inputs' may not be passed into 'reference-type inputs'."

Functions can receive array and variable parameters by reference or by value. However, if a function receives a variable or array by value, it is not possible to pass the parameter to a second function by reference. If an input of a function needs to be passed by reference to another function, it must also be declared as a reference input.

325 "Only an array, variable, or reference-input is allowed here"

Functions can receive arrays as parameters. If a function is expecting an array, any other type of parameter (variable, input, or reserved word) will display this error. In the following example:

```
Variable: MyVar(0);
Value1 = Average_a(MyVar, 10);
```

the function *Average_a()* requires an array on which to calculate an average and displays this error because *MyVar* is not an array.

Instead, you can write:

```
Array: MyArray[20](0);
Value1 = Average_a(MyArray, 10);
```

340 "This word is only allowed when defining array-type inputs."

This error is displayed when creating a function input using any input-type (such as *NumericArray*, *NumericArrayref*) without fully qualifying the input with braces. For example, this creates an error:

```
Input: MyInput(StringArrayRef);
```

because it does not include the array length parameter in brackets after the array name. The correct syntax would be:

```
Input: MyInput[n](StringArrayRef);
```

341 "An array input word (NUMERICARRAY, STRINGARRAY, TRUEFALSEARRAY, NUMERICARRAYREF, STRINGARRAYREF, TRUEFALSEARRAYREF) was expected here."

When declaring inputs that are meant to receive an array, one of the above words is expected as the input type. For example, this error will be displayed when declaring an input for a function using the following statement:

```
Input: MyArray[M,N](Numeric);
```

since the reserved word *Numeric* is not valid for declaring arrays. However, the following will verify successfully:

```
Input: MyArray[M,N](NumericArray);
```

342 "This word can only be used in a PaintBar study."

This error occurs when you use the reserved word *PlotPaintBar* when writing anything other than a PaintBar study.

396 "This statement cannot specify an odd number of plots."

This error is displayed when using the *PlotPaintBar* statement and specifying an odd number of plots. There are two possible uses for this statement, either specifying only a high and low value, or specifying high, low, open, and close markers. The correct syntax for the *PlotPaintBar* statement follows:

```
PlotPaintBar(High, Low, "PB");
```

or

```
PlotPaintBar(High, Low, Open, Close, "PB");
```

403 "Cannot implicitly convert String to Numerical"

Whenever the PowerEditor expects a numerical expression, and, instead, finds a text string expression, it will highlight the text string expression and display this message.

For example, the following statement will produce this error:

```
Variable: MyNumber("55");
Value1 = Close + MyNumber;
```

Instead, the following expression accomplishes the expected result because it first uses the keyword *StrToNum()* to convert a text string expression to a numeric value:

```
Variable: MyNumber("55");
Value1 = Close + StrToNum(MyNumber);
```

404 "Cannot implicitly convert String to TrueFalse"

Whenever the PowerEditor expects a true-false expression and, instead, finds a text string expression, it will highlight the text string expression and will display this message.

For example, the following statement will produce the error:

```
Input: Text1("Yes"), Text2("No");

Condition1 = Text1;
```

because the input "Text1" was declared as a text value and cannot be assigned the true-false variable *Condition1*. Instead, the following statement is correct:

```
Input: Text1("Yes"), Text2("No");
Condition1 = (Text1 = Text2);
```

Notice that while both *Text1* and *Text2* are string values, the result of the comparison is a true-false value which is properly assigned to a true-false variable.

405 "Cannot implicitly convert TrueFalse to String"

Whenever the PowerEditor expects a text string expression and, instead, finds a true-false expression, it will highlight the true-false expression and display this message. In this example, *Condition1* is a true-false variable and cannot be directly combined with a string:

```
FileAppend("Output.txt", "This is a text string" + Condition1);
```

Instead, the following expression corrects the problem by creating a string value based on whether *Condition1* is true or false:

```
Variable: txt(" ");

If Condition1 Then
   txt = "true"
Else
   txt = "false";
FileAppend("Output.txt", "This is a text string" + txt);
```

406 "Cannot implicitly convert Numerical to String"

Whenever the PowerEditor expects a text string expression and, instead, finds a numerical expression, it will highlight the numerical expression and will display this message.

For example:

```
FileAppend("Output.txt", "This is text" + Value1);
```

displays an error when a numeric expression is found. Instead, the following expression will accomplish the expected results because it uses the keyword *NumToString()* to convert a numerical expression to a string:

```
FileAppend("Output.txt", "This is text" + NumToStr(Value1, 2));
```

407 "Cannot implicitly convert TrueFalse to Numerical"

Whenever the PowerEditor expects a numerical value and, instead, finds a true-false expression, it will highlight the expression and will display this message.

For example, the following statement will produce this error because the *Condition1* value is a true-false variable and cannot be assigned to the numeric variable *Value1*:

```
Value1 = Condition1;
```

408 "Cannot implicitly convert Numerical to TrueFalse"

Whenever the PowerEditor expects a true/false expression and, instead, finds a numerical expression, it will highlight the numerical expression and will display this message. For example, the following statement produces an error because the reserved word *Open* is a numeric value and not a true/false expression:

```
Condition1 = Open;
```

Instead, assign the numeric value *Open* to the numeric variable *Value1*:

```
Value1 = Open ;
```

Or, change the statement such that it is a comparison. For example:

```
Condition1 = Open > Close;
```

Notice that while both *Open* and *Close* are numerical values, the result of the comparison is a true/false value, which is properly assigned to a true/false variable.

409 "String expression expected here"

This error is displayed whenever the PowerEditor is expecting a string expression and, instead, it finds a numeric or true-false expression. For example, this error will be displayed when writing information to a file with a *FileAppend* statement:

```
FileAppend("file.txt", Value1);
```

that includes the numeric expression *Value1* instead of a text string. Numeric expressions can be converted to strings by using the *NumToStr()* keyword. For example:

```
FileAppend("file.txt", NumToStr(Value1,2));
```

569 "Buy or SellShort name within double quotes expected here."

When specifying the name of a trading strategy, only a text string literal can be used, and it can't be substituted by a variable or an input. The following statements will generate this error:

```
Variable: txt("MyStrategy");

Buy (txt) Next Bar at Market;
```

while the correct way of assigning a name to a strategy is to use a literal string, such as:

```
Buy ("Strategy Name") Next Bar at Market;
```

APPENDIX

B

TradeStation 2000i Source Code of Select Programs

Bollinger Bandit {TradeStation 2000i Format}

```
Vars: upBand(0),dnBand(0),liqDays(50);

upBand = BollingerBand(Close,50,1.25);
dnBand = BollingerBand(Close,50,–1.25);

if(MarketPosition <> 1 and ExitsToday(date) = 0) then Buy("BanditBuy")tomorrow upBand
  stop;
if(MarketPosition <> –1 and ExitsToday(date) = 0) then Sell("BanditSell")tomorrow dnBand
  stop;

if(MarketPosition = 0) then liqDays = 50;
if(MarketPosition <> 0) then
begin
       liqDays = liqDays – 1;
       liqDays = MaxList(liqDays,10);
end;
if(MarketPosition = 1 and Average(Close,liqDays) < upBand) then ExitLong("LongLiq")
tomorrow Average(Close,liqDays)stop;
if(MarketPosition = –1 and Average(Close,liqDays) > dnBand) then ExitShort("ShortLiq")
tomorrow Average(Close,liqDays)stop;
```

{Dynamic Break Out II by George Pruitt—TradeStation 2000i Format

```
This system is an extension of the original Dynamic Break Out system written by George
for Futures Magazine in 1996. In addition to the channel break out methodology, DBS II
incorporates Bollinger Bands to determine trade entry.}

Inputs: ceilingAmt(60),floorAmt(20),bolBandTrig(2.00);
Vars: lookBackDays(20),todayVolatility(0),yesterDayVolatility(0),deltaVolatility(0);
Vars: buyPoint(0),sellPoint(0),longLiqPoint(0),shortLiqPoint(0),upBand(0),dnBand(0);

todayVolatility = StdDev(Close,30);
yesterDayVolatility = StdDev(Close[1],30); {See how I offset the function call to
                                            get yesterday's value}
deltaVolatility = (todayVolatility – yesterDayVolatility)/todayVolatility;
lookBackDays = lookBackDays * (1 + deltaVolatility);
lookBackDays = Round(lookBackDays,0);
lookBackDays = MinList(lookBackDays,ceilingAmt); {Keep adaptive engine within bounds}
lookBackDays = MaxList(lookBackDays,floorAmt);



upBand = BollingerBand(Close,lookBackDays,+BolBandTrig);
dnBand = BollingerBand(Close,lookBackDays,–BolBandTrig);

buyPoint = Highest(High,lookBackDays);
sellPoint = Lowest(Low,lookBackDays);

longLiqPoint = Average(Close,lookBackDays);  {Exit long at 1/2 look back period}
shortLiqPoint = Average(Close,lookBackDays); {Exit short at 1/2 look back period}

if(Close > upBand) then Buy("DBS-2 Buy") tomorrow at buyPoint stop;
if(Close < dnBand) then Sell("DBS-2 Sell") tomorrow at sellPoint stop;

if(MarketPosition = 1) then ExitLong("LongLiq") tomorrow at longLiqPoint stop;
if(MarketPosition = –1) then ExitShort("ShortLiq") tomorrow at shortLiqPoint stop;
```

{DBS II Fade by George Pruitt—TradeStation 2000i

```
This version of the DBS buys when the original DBS sold and sells when the original DBS
bought. We did this to illustrate the seasonal/cyclical nature of the soybeans}

Inputs: ceilingAmt(60),floorAmt(20),bolBandTrig(2.00);
Vars: lookBackDays(20),todayVolatility(0),yesterDayVolatility(0),deltaVolatility(0);
Vars: buyPoint(0),sellPoint(0),longLiqPoint(0),shortLiqPoint(0),upBand(0),dnBand(0);

todayVolatility = StdDev(Close,30);
yesterDayVolatility = StdDev(Close[1],30); {See how I offset the function call to
                                            get yesterday's value}
deltaVolatility = (todayVolatility – yesterDayVolatility)/todayVolatility;
lookBackDays = lookBackDays * (1 + deltaVolatility);
lookBackDays = Round(lookBackDays,0);
lookBackDays = MinList(lookBackDays,ceilingAmt); {Keep adaptive engine within bounds}
lookBackDays = MaxList(lookBackDays,floorAmt);


upBand = BollingerBand(Close,lookBackDays,+bolBandTrig);
dnBand = BollingerBand(Close,lookBackDays,–bolBandTrig);

buyPoint = Highest(High,lookBackDays);
sellPoint = Lowest(Low,lookBackDays);

longLiqPoint = Average(Close,lookBackDays);  {Exit long at 1/2 look back period}
shortLiqPoint = Average(Close,lookBackDays); {Exit short at 1/2 look back period}

if(Close > upBand) then Sell("DBS-2 Buy") tomorrow at buyPoint limit;
if(Close < dnBand) then Buy("DBS-2 Sell") tomorrow at sellPoint limit;

if(MarketPosition = 1) then ExitLong("LongLiq") tomorrow at longLiqPoint limit;
if(MarketPosition = –1) then ExitShort("ShortLiq") tomorrow at shortLiqPoint limit;
```

{King Keltner Program

```
King Keltner by George Pruitt—based on trading system presented by Chester Keltner—
an example of a simple, robust and effective strategy.}

Inputs: avgLength(40),atrLength(40);
Vars: upBand(0),dnBand(0),liquidPoint(0),movAvgVal(0);

movAvgVal = average((High + Low + Close)/3.0,avgLength);

upBand = movAvgVal + AvgTrueRange(atrLength);
dnBand = movAvgVal — AvgTrueRange(atrLength);

{Remember buy stops are above the market and sell stops are below the market—
if the market gaps above the buy stop, then the order turns into a market order
vice versa for the sell stop}

if(movAvgVal > movAvgVal[1]) then Buy ("KKBuy") tomorrow at upBand stop;
if(movAvgVal < movAvgVal[1]) then Sell("KKSell")tomorrow at dnBand stop;

liquidPoint = movAvgVal;

if(MarketPosition = 1) then ExitLong tomorrow at liquidPoint stop;
if(MarketPosition = —1) then ExitShort tomorrow at liquidPoint stop;
```

{MyAdxSys—TradeStation 2000i}

```
Inputs: adxLength(14),mavLength(9),mavLength2(19);
Vars:adxVal(0);

adxVal = Adx(adxLength);

if(adxVal>=15) then
begin
       if(Average(Close,mavLength1) crosses above Average(Close,mavLength2)) then
              buy tomorrow at High stop;
       if(Average(Close,mavLength1) crosses below Average(Close,mavLength2)) then
              sell tomorrow at Low stop;
end;
if(adxVal<15) then
begin
       if(MarketPosition = 1) then ExitLong next bar at Lowest(Low,4) stop;
       if(MarketPosition = –1) then ExitShort next bar at Highest(High,4) stop;
end;
```

{MyMomRsi—TradeStation 2000i Format

```
Combines Momentum and the RSI into one strategy and incorporates built-in protective stop
and trailing stop functions.}

Inputs: momLength(14),rsi(14),protStop$(3000),
        trailStopThresh$(3000),trailStopPrcnt(25);

if(Momentum(Close,momLength)>0 and
      RSI(Close,rsiLength) crosses below 60) then
      {crosses below means the same as
      RSI(Close,rsiLength)[1]>60 and RSI(Close,rsiLength)<60}
      begin
            Buy("Mom+RetB")next bar at Lowest(Low,3) limit;
      end;
if(Momentum(Close,momLength)<0 and
      RSI(Close,rsiLength) crosses above 40) then
      begin
            Sell("Mom—RetS") next bar at Highest(High,3) limit;
      end;

SetStopLoss(protStop$);
SetPercentTrailing(trailStopThresh$,trailStopPrcnt);
```

{MyMovAvgSys—TradeStation 20001 Format

```
demonstrates the use of nested function calls.}

Inputs: movAvgLength1(9),movAvgLength2(19),channelLength(20);

value1 = Highest(Average(Close,movAvgLength1),channelLength);
value2 = Lowest(Average(Close,movAvgLength1),channelLength);

Buy tomorrow at value1 stop;
Sell tomorrow at value2 stop;

{MyStrategy-1 or MySignal-1—TradeStation 2000i Format
The hello world program of EasyLanguage}

Inputs: longLength(40),shortLength(40);

Buy tomorrow at Highest(High,longLength)stop;
Sell tomorrow at Lowest(Low,shortLength)stop;
```

{MyTrailPrcntStop—TradeStation 2000i Format

```
user defined/programmed percent trailing stop.}

Inputs: trailStopThresh(3000),trailStopPrcnt(25);
Vars: maxPositionProf(0),longLiqPoint(0),prevMarketPosition(0);

if(marketposition <> 1) then buy tomorrow at Highest(high,10) on stop;


if(marketposition <> prevMarketPosition) then maxPositionProf = 0;
prevMarketPosition = MarketPosition;
if(marketPosition = 1) then
begin
       maxPositionProf = maxlist(High–entryPrice,maxPositionProf);
  if(maxPositionprof*bigPointValue>trailStopThresh) then
       begin
              longLiqPoint = EntryPrice + (maxPositionprof*(1–trailStopPrcnt/100);
              ExitLong next bar longLiqPoint stop;
       end;
end;
SetStepLoss(3000);
```

{Seasonal Soybean—TradeStation 2000i Format}

```
Inputs: goLongStart(301),goLongEnd(701),goShortStart(702),goShortEnd(228);
Vars: monthAndDay(0);

{The inputs represent the months and days that we can enter long and short trades}
{301 is March 01   >> can only go long from this date and up to
701 is July 01   >> this date
702 is July 02   >> can only go short from this date and up to
228 is February 28 >> this date}


{Let's use the date and extract the information that we need to determine the month and
day}
{If we devide the date by 10000, the remainder is the month and day. We can use the
modulus function which determines the remainder of division}
monthAndDay = Mod(Date of tomorrow,10000);
if(monthAndDay >= goLongStart and monthAndDay <= goLongEnd) then
begin
       buy("Seasonal Buy") tomorrow at Open;
end;
if(monthAndDay >= goShortStart or monthAndDay <= goShortEnd) then
{Notice that we had to use "or" instead of "and"—this is due
to the goShortEnd date is less than the goShortStart date}
begin;
        sell("Seasonal Sell") tomorrow at Open;
end;
```

{Super Combo by George Pruitt

```
This intraday trading system will illustrate the multiple data handling capabilities of
TradeStation. All pertinent buy and sell calculations will be based on daily bars and
actual trades will be executed on 5-min bars. I have made most of the parameters input
variables.}

Inputs:waitPeriodMins(30),initTradesEndTime(1430),liqRevEndTime(1200),
       thrustPrcnt1(0.30),thrustPrcnt2(0.60),breakOutPrcnt(0.25),
       failedBreakOutPrcnt(0.25),protStopPrcnt1(0.25),protStopPrcnt2(0.15),
       protStopAmt(3.00),breakEvenPrcnt(0.50),avgRngLength(10),avgOCLength(10);

Variables:averageRange(0),averageOCRange(0),canTrade(0),buyEasierDay(FALSE),
       sellEasierDay(FALSE),buyBOPoint(0),sellBOPoint(0),longBreakPt(0),
       shortBreakPt(0),longFBOPoint(0),shortFBOPoint(0),barCount(0),
       intraHigh(0),intraLow(999999),buysToday(0),sellsToday(0),
       currTrdType(0),longLiqPoint(0),shortLiqPoint(0),yesterdayOCRRange(0),
       intraTradeHigh(0),intraTradeLow(999999);


{Just like we did in the pseudocode—lets start out with the daily
bar calculations. If Date <> Date[1]—first bar of day}
if(Date <> Date[1]) then {save time by doing these calculations once per day}
begin
       averageRange = Average(Range,10) of Data2; {Data 2 points to daily bars}
       yesterdayOCRRange = AbsValue(Open of Data2—Close of Data2);
       average OCRange = Average(AbsValue(Open of Data2—Close of Data2),10);
       print(date,time,yesterdayOCRRange,averageOCRange);
       canTrade = 0;
       if(yesterdayOCRRange<0.80*averageOCRange) then canTrade = 1;
       buyEasierDay = FALSE;
       sellEasierDay = FALSE;

       if(Close of Data2 <= CLose[1] of Data2) then buyEasierDay = TRUE;
       if(Close of Data2 > Close[1] of Data2) then sellEasierDay = TRUE;

       if(buyEasierDay) then
       begin
              buyBOPoint = Open of data1 + thrustPrcnt1*averageRange;
              sellBOPoint = Open of data 1 — thrustPrcnt2*averageRange;
       end;
       if(sellEasierDay) then
       begin
              sellBOPoint = Open of data1 — thrustPrcnt1*averageRange;
              buyBOPoint = Open of data1 + thrustPrcnt2*averageRange;
       end;
       longBreakPt = High of Data2 + breakOutPrcnt*averageRange;
       shortBreakPt = Low of Data2 — breakOutPrcnt*averageRange;
```

```
        shortFBOPoint = High of Data2 – failedBreakOutPrcnt*averageRange;
        longFBOPoint = Low of Data2 + failedBreakOutPrcnt*averageRange;

{Go ahead and initialize any variables that we may need later on in the day}

        barCount = 0;
        intraHigh = 0;intraLow = 999999;{Didn't know you could do this}
        buysToday = 0;sellsToday = 0;{You can put multiple statements on one line}

        currTrdType = 0;
end;
{Now lets trade and manage on 5-min bars}
if(High > intraHigh) then intraHigh = High;
if(Low < intraLow) then intraLow = Low;

barCount = barCount + 1; {count the number of bars of intraday data}
if(barCount >= waitPeriodMins/BarInterval and canTrade = 1) then {have we waited long
enough}
begin
        if(MarketPosition = 0) then
        begin
                intraTradeHigh = 0;
                intraTradeLow = 999999;
        end;

        if(MarketPosition = 1) then
        begin
                intraTradeHigh = MaxList(intraTradeHigh,High);
                buysToday = 1;
        end;
        if(MarketPosition = –1) then
        begin
                intraTradeLow = MinList(intraTradeLow,Low);
                sellsToday = 1;
        end;

        if(buysToday = 0 and Time < initTradesEndTime) then
                Buy("LBreakOut") next bar at buyBOPoint stop;
        if(sellsToday = 0 and Time < initTradesEndTime) then
                Sell("SBreakOut") next bar at sellBOPoint stop;
        if(intraHigh > longBreakPt and sellsToday = 0 and Time < initTradesEndTime) then
                Sell("SfailedBO") next bar at shortFBOPoint stop;
        if(intraLow < shortBreakPt and buysToday = 0 and Time < initTradesEndTime) then
                Buy("BfailedBO") next bar at longFBOPoint stop;
{The next module keeps track of positions and places protective stops}
        if(MarketPosition = 1) then
        begin
                longLiqPoint = EntryPrice–protStopPrcnt1*averageRage;
                longLiqPoint = MinList(longLiqPoint,EntryPrice – protStopAmt);
                if(MarketPosition(1) = –1 and BarsSinceEntry = 1 and
```

```
                    High[1] >= shortLiqPoint and shortLiqPoint < shortFBOPoint) then
                          currTrdType = –2; {we just got long from a short liq reversal}
             if(currTrdType = –2) then
             begin
                    longLiqPoint = EntryPrice – protStopPrcnt2*averageRange;
                    longLiqPoint = MinList(longLiqPoint,EntryPrice – protStopAmt);
             end;
             if(intraTradeHigh >= EntryPrice + breakEvenPrcnt*averageRange) then
                    longLiqPoint = EntryPrice;   {Breakeven trade}
             if(Time >= initTradesEndTime) then
                    longLiqPoint = MaxList(longLiqPoint,Lowest(Low,3)); {Trailing stop}
             if(Time < liqRevEndTime and sellsToday = 0 and
             longLiqPoint <> EntryPrice and BarsSinceEntry > 4) then
             begin
                    Sell("LongLiqRev") next bar at longLiqPoint stop;
             end
             else begin
                    ExitLong("LongLiq") next bar at longLiqPoint stop;
             end;
       end;
       if(MarketPosition = –1) then
       begin

             shortLiqPoint = EntryPrice+protStopPrcnt1*averageRange;
             shortLiqPoint = MaxList(shortLiqPoint,EntryPrice + protStopAmt);
             if(MarketPosition(1) = 1 and BarsSinceEntry(0) = 1 and
                    Low [1] <= longLiqPoint and longLiqPoint > longFBOPoint) then
                          currTrdType = +2; {we just got long from a short liq reversal}
             if(currTrdType = +2) then
             begin
                    shortLiqPoint = EntryPrice + protStopPrcnt2*averageRange;
                    shortLiqPoint = MaxList(shortLiqPoint,EntryPrice + protStopAmt);
             end;
             if(intraTradeLow <= EntryPrice – breakEvenPrcnt*averageRange) then
                    shortLiqPoint = EntryPrice;  {Breakeven trade}
             if(Time >= initTradesEndTime) then
                 shortLiqPoint = MinList(shortLiqPoint,Highest(High,3)); {Trailing stop}
             if(Time < liqRevEndTime and buysToday = 0 and
             shortLiqPoint <> EntryPrice and BarsSinceEntry > 4) then
             begin
                    Buy("ShortLiqRev") next bar at shortLiqPoint stop;
             end
             else begin
                    ExitShort("ShortLiq") next bar at shortLiqPoint stop;
             end;
       end;
end;
SetExitOnClose;
```

{Thermostat by George Pruitt

```
Two systems in one. If the ChoppyMarketIndex is less than 20
then we are in a swing mode. If it is greater than or equal
to 20 then we are in a trend mode. Swing system is an open range
breakout incorporating a buy easier/sell easier concept. The trend
following system is based on bollinger bands and is similar to the

BollingerBandit program.}

Inputs: bollingerLengths(50),trendLiqLength(50),numStdDevs(2),
        swingPrcnt1(0.50),swingPrcnt2(0.75),atrLength(10),
        swingTrendSwitch(20);
Vars:cmiVal(0),buyEasierDay(0),sellEasierDay(0),trendLokBuy(0),
        trendLokSell(0),keyOfDay(0),swingBuyPt(0),swingSellPt(0),
        trendBuyPt(0),trendSellPt(0),swingProtStop(0);

cmiVal = ChoppyMarketIndex(30);


buyEasierDay = 0;
sellEasierDay = 0;

trendLokBuy = Average(Low,3);
trendLokSell = Average(High,3);

keyOfDay = (High + Low + Close)/3;
if(Close > keyOfDay) then sellEasierDay = 1;
if(Close <= keyOfDay) then buyEasierDay = 1;

if(buyEasierDay = 1) then
begin
       swingBuyPt = Open of tomorrow + swingPrcnt1*AvgTrueRange(atrLength);
       swingSellPt = Open of tomorrow – swingPrcnt2*AvgTrueRange(atrLength);
end;
if(sellEasierDay = 1) then
begin
       swingBuyPt = Open of tomorrow + swingPrcnt2*AvgTrueRange(atrLength);
       swingSellPt = Open of tomorrow – swingPrcnt1*AvgTrueRange(atrLength);
end;

swingBuyPt = MaxList(swingBuyPt,trendLokBuy);
swingSellPt = MinList(swingSellPt,trendLokSell);

trendBuyPt = BollingerBand(Close,bollingerLengths,numStdDevs);
trendSellPt = BollingerBand(Close,bollingerLengths,–numStdDevs);

if(cmiVal < swingTrendSwitch) then
```

```
begin
       if(MarketPosition <> 1) then Buy("SwingBuy") next bar at swingBuyPt stop;
       if(MarketPosition <> –1) then Sell("SwingSell") next bar at swingSellPt stop;
end
else
begin
       swingProtStop = 3*AvgTrueRange(atrLength);
       Buy("TrendBuy") next bar at trendBuyPt stop;
       Sell("TrendSell") next bar at trendSellPt stop;
       ExitLong from Entry("TrendBuy") next bar at Average(Close,trendLiqLength) stop;
       ExitShort from Entry("TrendSell") next bar at Average(Close,trendLiqLength) stop;
       ExitLong from Entry("SwingBuy") next bar at EntryPrice – swingProtStop stop;
       ExitShort from Entry("SwingSell") next bar at EntryPrice + swingProtStop stop;
end;
```

{The Ghost Trader by George Pruitt TradeStation 2000i Format}

```
This strategy keeps track of the simulated trades of a trading system and
makes real buying and selling decisions based on the performance of the
simulated system}

vars: myPosition(0),myEntryPrice(0),myProfit(-1);

{Ghost system }

{Look to see if a trade would have been executed today and keep track
of our position and our entry price. Test today's high/low price
against the trade signal that was generated by offsetting our calculations
by one day.}

if(myPosition = 0 and Xaverage(Close[1],9) > Xaverage(High[1],19) and
       RSI(Close[1],9) crosses below 70 and High >= High[1]) then
begin
       myEntryPrice = MaxList(Open,High[1]); {Check for a gap open}
       myPosition = 1;
end;
if(myPosition = 1 and Low < Lowest(Low[1],20) )then
begin
       value1 = MinList((Lowest(low[1],20)),Open); {Check for a gap open}
       myProfit = value1 - myEntryPrice;           {Calculate our trade profit/loss}
       myPosition = 0;
end;

if(myPosition = 0 and Xaverage(Close[1],9) < Xaverage(Low[1],19) and
       RSI(Close[1],9) crosses above 30 and Low <= Low[1]) then
begin
       myEntryPrice = MinList(Open,Low[1]);
       myPosition = -1;
end;

if(myPosition = -1 and High > Highest(High[1],20)) then
begin
       value1 = MaxList((Highest(High[1],20)),Open);{Check again for a gap open}
       myProfit = myEntryPrice - value1;            {Calculate our trade profit/loss}
       myPosition = 0;
end;
```

```
{Real System}

{Only enter a new position if the last simulated or real trade was a loser.
If last trade was a loser, myProfit will be less than zero.}

if(marketPosition = 0 and myProfit < 0 and Xaverage(Close,9) > Xaverage(High,19) and
RSI(Close,9) crosses below 70) then
begin
       Buy next bar at High stop;
end;
if(marketPosition = 0 and myProfit < 0 and Xaverage(Close,9) < Xaverage(Low,19) and
RSI(Close,9) crosses above 30) then
begin
       Sell next bar at Low stop;
end;

if(marketPosition = 1) then Exit Long next bar at Lowest(Low,20) stop;
if(marketPosition = -1) then Exit Short next bar at Highest(High,20) stop;
```

{The Money Manager by George Pruitt TradeStation 2000i Format}

```
{Demonstrates the programming and use of a money management scheme.}
{The user inputs initial capital and the amount he wants to risk on each trade.}
Inputs: initCapital(100000),rskAmt(.02);
Vars: marketRisk(0),numContracts(0);

marketRisk = StdDev(Close,30) * BigPointValue;

numContracts = (initialCapital * rskAmt) / marketRisk;

value1 = Round(numContracts,0);
if(value1 > numContracts) then
numContracts = value1 – 1
else
numContracts = value1;

numContracts = MaxList(numContracts,1); {make sure at least 1 contract is traded}

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

if(MarketPosition = 1) then ExitLong("LongLiq") next bar at Lowest(Low,20) stop;
if(MarketPosition = -1) then ExitShort("ShortLiq") next bar at Highest(High,20) stop;
```

APPENDIX

C

Reserved Words Quick Reference

#BEGINALERT

A compiler directive that executes instructions between **#BeginAlert** and **#End** only when the **Enable Alert** check box is selected.

Usage:
```
#BeginAlert
        Alert("ADX Alert");
#End;
```

#BEGINCMTRY

A compiler directive that executes instructions between **#BeginCmtry** and **#End** only when using the **Analysis Commentary** tool to select a bar on a chart or a cell on a grid.

Usage:
```
#BeginCmtry
        Commentary("The value is " + NumtoStr(Plot1, 0));
#End;
```

#BEGINCMTRYORALERT

A compiler directive that executes instructions between **#BeginCmtryOrAlert** and **#End** when either the Alert or Commentary conditions exist.

Usage:
```
#BeginCmtryorAlert
        Alert("ADX Alert");
        Commentary("The value is " + NumtoStr(Plot1, 0));
#End;
```

#END

A compiler directive used to terminate an alert or commentary block statement.

AB_AddCell

Adds a cell to an ActivityBar row.

Syntax: **AB_AddCell**(*Price*, *Side*, *Str_Char*, *Color*, *Value);*

Price: a numeric expression representing the price of a bar (e.g.,Open,Close)
Side: **LeftSide**,**RightSide**
Str_Char: a character that is displayed in the ActivityBar cell (e.g.,"A","N")
Color: an EasyLanguage color value (e.g.,Red, Black)
Value: a numeric expression representing the value of the cell

Usage:
```
AB_AddCell(Open, Leftside, "A", Red, 1) ;
```

AB_AddCellRange

Adds cells to a price range of the current bar starting at LowValue to HighValue.

Syntax: **AB_AddCellRange**(*RangeHi*, *RangeLo*, *Side*, *Label*, *Color*, *Value*)

RangeHi: a numeric expression representing the highest *price* for a column
RangeLo: a numeric expression representing the lowest *price* for a column
Side: **LeftSide**,**RightSide**
Label: a character that will be placed in the ActivityBar cell (e.g.,"A","N")
Color: an EasyLanguage color value (e.g.,Red, Black)
Value: a numeric expression representing the value of each cell to be added

Usage:
```
Value1 = AB_AddCellRange(High of ActivityData, Low of
ActivityData, RightSide, "U", Green, 0);
```

AB_AverageCells

Returns the average number of ActivityBar cells per row for the current bar.

Syntax: **AB_AverageCells**(*Side*)

Side: **LeftSide**,**RightSide**

Usage:
```
Value2 = AB_AverageCells(RightSide);
```

AB_AveragePrice

Returns the average price of the ActivityBar cells on one or both sides.

Syntax: **AB_AveragePrice**(*Side*)

Side: **LeftSide**,**RightSide**

Usage:
```
Value2 = AB_AveragePrice(LeftSide);
```

AB_CellCount

Counts and returns the number of cells on one or both sides of an ActivityBar.

Syntax: **AB_CellCount**(*Side*)

Side: **LeftSide**,**RightSide**

Usage:
```
Value2 = AB_CellCount(LeftSide);
```

AB_GetCellChar

Returns the text string expression stored in the specified cell.

Syntax: **AB_GetCellChar**(*Price*,*Side*,*Column*)
Price: price value of the row containing the character
Side: **LeftSide**,**RightSide**
Column: number of the cell column containing the character on the side specified

Usage:
```
Str = AB_GetCellChar(Close, RightSide, 3) ;
```

AB_GetCellColor

Returns the color of the character stored in the specified cell.

Syntax: **AB_GetCellColor**(*Price*,*Side*,*Column*)
Same parameters as AB_GetCellChar above.

Usage:
```
Value1 = AB_GetCellChar(Open, LeftSide, 2) ;
```

AB_GetCellDate

Returns the corresponding date of the specified cell.

Syntax: **AB_GetCellDate**(*Price*,*Side*,*Column*)
Same parameters as AB_GetCellChar above.

Usage:
```
Value2 = AB_GetCellDate(High,RightSide, 5) ;
```

AB_GetCellTime

Returns the corresponding time of the specified cell.

Syntax: **AB_GetCellTime**(*Price*,*Side*,*Column*)
Same parameters as AB_GetCellChar above.

Usage:
```
Value1 = AB_GetCellTime(Low,LeftSide, 4) ;
```

AB_GetCellValue

Returns the extra value stored in the specified cell.

Syntax: **AB_GetCellValue**(*Price*,*Side*,*Column*)
Same parameters as AB_GetCellChar above.

Usage:
```
Value2 = AB_GetCellValue(High,RightSide, 1) ;
```

AB_GetNumCells

Returns how many cells exist at a specified price on the right or left side.

Syntax: **AB_GetNumCells**(*Price*,*Side*)
Price: price value of the row
Side: **LeftSide**,**RightSide**

Usage:
```
Value1 = AB_GetNumCells(Close,LeftSide) ;
```

AB_GetZoneHigh

Returns the value of the top (high) of the ActivityBar zone.

Syntax: **AB_GetZoneHigh**(*Side*)
Side: **LeftSide**,**RightSide**

Usage:
```
Value1 = AB_GetZoneHigh(LeftSide) ;
```

AB_GetZoneLow

Returns the value of the bottom (low) of the ActivityBar zone.

Syntax: **AB_GetZoneLow**(*Side*)
Side: **LeftSide**,**RightSide**

Usage:
```
Value2 = AB_GetZoneLow(RightSide) ;
```

AB_High

Returns the high of the current ActivityBar.

Usage:
```
Value1 = AB_High ;
```

AB_Low

Returns the low of the current ActivityBar.

Usage:
```
Value1 = AB_Low ;
```

AB_Median

Returns the median price value of the cells for the current ActivityBar.

Syntax: **AB_Median**(*Side*)
Side: **LeftSide**,**RightSide**

Usage:
```
Value2 = AB_Median(RightSide) ;
```

AB_Mode

Returns the cell count of the row with the most cells (the Mode row) and the price of the Mode row.

Syntax: **AB_Mode**(*Side*, *Type*, *oModeCount*, *oModePrice*)
Side: **LeftSide**,**RightSide**
Type: >= 0 for Largest mode, < 0 for smallest mode
oModeCount: Variable or array element that takes the number of cells (passed by reference)
oModePrice: Variable or array element that takes the Mode price (passed by reference)

Usage:
```
Value1 = AB_ModeCount(LeftSide) ;
```

AB_NextColor

Specifies the color of ActivityBar cells based on a user-defined interval.

Syntax: **AB_NextColor**(*MinuteInterval*)
MinuteInterval: number of minutes that make up each cell color interval

Usage:
```
Value1 = AB_NextColor(10);
```

AB_NextLabel

Returns a letter/number to use in an ActivityBar cell based on a user-defined interval.

Syntax: **AB_NextLabel**(*MinuteInterval*)
MinuteInterval: number of minutes that make up each cell label interval

Usage:
```
Value1 = AB_NextLabel(10);
```

AB_RemoveCell

Removes a cell from an ActivityBar row.

Syntax: **AB_RemoveCell**(*Price*,*Column*,*Side*)
Price: price value of the cell to remove
Column: number of the column containing the cell on the side specified
Side: **LeftSide**,**RightSide**

Usage:
```
Value1 = AB_RemoveCell(Close,3,RightSide) ;
```

AB_RowHeight

Returns the row (cell) height for an ActivityBar. Often used with **AB_SetRowHeight**.

Usage:
```
Value1 = AB_RowHeight ;
```

AB_RowHeightCalc

Calculates and returns the row height to use for an ActivityBar.

Syntax: **AB_RowHeightCalc**(*ApproxNumRows*,*RangeAvgLength*)
ApproxNumRows: the approximate number of rows desired (usually between 5 and 25)
RangeAvgLength: number of bars back used to determine the average price range

Usage:
```
Value2 = AB_RowHeightCalc(10, 5) ;
```

AB_SetActiveCell

Changes the placement of the ActivityBar marker to the specified location on the bar.

Syntax: **AB_SetActiveCell**(*Price*,*Side*)
Price: price value of the cell row
Side: **LeftSide**,**RightSide**

Usage:
```
AB_SetActiveCell(Open,RightSide) ;
```

AB_SetRowHeight

Changes the current ActivityBar's row-increment value.

Syntax: **AB_SetRowHeight**(*RowHeight*)
RowHeight: value representing the row spacing for cells. Generally use AB_RowHeightCalc as the parameter.

Usage:
```
AB_SetRowHeight(AB_RowHeightCalc(10,5)) ;
```

AB_SetZone

Sets a zone range box for an ActivityBar side.

Syntax: **AB_SetZone**(*HighPrice*,*LowPrice*, *Side*)
HighPrice: a numeric expression representing the high *price* of the zone range box
LowPrice: a numeric expression representing the low *price* of the zone range box
Side: **LeftSide**,**RightSide**

Usage:
```
AB_SetZone(Average(High, 5), Average(Low, 5), RightSide);
```

AB_StdDev

Returns the standard deviation of the ActivityBar cells for the specified side.

Syntax: **AB_StdDev**(*Multiplier*, *Side*)
Multiplier: represents the number of standard deviations to calculate
Side: ***LeftSide***,***RightSide***,***Both***

Usage:
```
Value2 = AB_StdDev(2, LeftSide);
```

Above

Used only with **Crosses** to detect a value crossing above, or over, another value.

Usage:
```
If Plot11 Crosses Above Plot2 Then {Any Operation} ;
```

AbsValue

Absolute value of num.

Syntax: **AbsValue**(*Num*)
Num: a numeric value or expression

Usage:
```
Value1 = AbsValue(-1.45);   {returns a value of 1.45}
```

ActivityData

References any bar data element (Open, upticks, etc.) of the ActivityBar.

Usage:
```
Value2 = AB_AddCellRange(High of ActivityData,Low of
ActivityData,Rightside,3,2);
```

AddToMovieChain

Appends movie file *MFile* to end of movie chain *MChain*.

Syntax: **AddToMovieChain**(*MFile,MChain*)
MFile: a numeric expression representing a movie chain ID
MChain: a string expression representing the path and name of the *.avi file to be added to the specified movie chain

Ago

References a specified number of bars back already analyzed by EasyLanguage.

Usage:
```
Value1 = Close of 1 Bar Ago;   {returns Close of the previous bar}
```

Alert

When *True*, triggers an alert for an indicator or study. The alert description is optional.

Usage:
```
If {Your Alert Criteria} Then Alert("MyAlert");
```

AlertEnabled

Returns *True* if the **Enable Alert** check box is selected.

Usage:
```
If AlertEnabled Then Begin
       {Your Code Here}
End ;
```

All

Specifies all shares/contracts are to be sold/covered when exiting a position.

Usage:
```
If Condition1 Then Sell All Shares Next Bar at Market;
```

An

Skip word used to improve readability. Ignored during execution.

Usage: `If an Open is > 100 Then` *{any operation}*

AND

Links 2 true/false expressions together. *True* if both expressions are true.

Usage:
```
If Plot1 Crosses Above Plot2 AND Plot2 > 5 Then
    {any operation};
```

Arctangent

Returns the arctangent value of *num* degrees.

Usage: `Value1 = Arctangent(45);` {returns 88.7270 when *num* is 45 degrees}

Array

Used to declare an array type of variable.

Syntax: **Array:** AnyName[Elements](InitialValue)

Elements: the number of indexed values that this array can store

InitialValue: a numeric expression used to set the initial value of each element

Usage: `Array: AnyName[4](0);` {declares a 4 element array with '0' for initial values}

Arrays

Used to declare an array type of variable.

See **Array**.

ARRAYSIZE

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

ARRAYSTARTADDR

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

At

Skip word used to improve readability. Ignored during execution.

Usage: `Buy 100 Contracts on Next Bar at Market;`

At$

Anchors exit prices to the bar where the named entry order was placed.

Usage: `Sell from Entry("MA Cross") At$ Low - 1 Point Stop;`

AtCommentaryBar

Returns *True* if the current bar was selected with the Analysis Commentary Tool.

Usage: `If AtCommentaryBar Then` *{your commentary} ;*

AvgBarsLosTrade

The average number of bars that elapsed during losing trades for all closed trades.

Usage: `Value1 = AvgBarsLosTrade;` {**Note**: returns the integer portion of the average}

AvgBarsWinTrade

The average number of bars that elapsed during winning trades for all closed trades.

Usage: `Value2 = AvgBarsWinTrade;` {**Note**: returns the integer portion of the average}

AvgEntryPrice

Returns the average entry price of each open entry in a pyramided position.

Usage: `Value1 = AvgEntryPrice;` {returns 70 for open trades entered at 45, 75 and 90}

AvgList

Returns the average of the listed values.

Usage: `Value2 = AvgList(18, 67, 98, 24, 65, 19);` {returns a value of 48.5}

Bar

References values for a specific bar based on the data interval.

Usage: `Buy Next Bar at Open ;`

BarInterval

Returns the data interval (in minutes) for bars on a minute-based chart.

Bars

References a bar occurring N bars ago based on the data interval.

Usage: `Value2 = Open of 5 Bars Ago ;`

BarsSinceEntry

Bars since initial entry of position, num position(s) ago.

Syntax: **BarsSinceEntry**(*Num*)

Num: number of positions ago, 0 for current position

BarsSinceExit

Bars since position closed-out, num position(s) ago.

Syntax: **BarsSinceExit**(*Num*)

Num: number of positions ago, 0 for current position

BarStatus

Determines if a trade (tick) opened the bar, closed the bar, or is 'inside the bar.'

Syntax: **BarsStatus**(*DataSeries*)

DataSeries: specifies which data series to use

Returns: 0 for opening tick, 1 for inside tick, 2 for closing tick, -1 on an error

Usage: `Value2 = BarStatus(2) ;`

BarType

The compression setting of the price data for the applied analysis technique.

Returns: 0 forTick, 1 for Intraday, 2 for Daily, 3 for Weekly, 4 for Monthly, 5 for Point & Figure.

Usage: `If BarType = 2 Then {Any Operation}` {tests for daily bars}

Based

Skip word retained for backward compatibility.

Begin

Used to begin a block of EasyLanguage instructions within a conditional statement.

Usage:
```
If Condition1 = True Then Begin
     {Your Code Line1}
     {Your Code Line2, etc.}
  End;
```

Below

Used only with **Crosses** to detect a value crossing below, or under, another value

Usage: `If Value1 Crosses Below Value2 Then {Any Operation} ;`

Beta

Returns the Beta value of a stock compared to the S&P 500 index.

Beta_Down

Returns the Beta value of a stock when S&P 500 is down.

Beta_Up

Returns the Beta value of a stock when S&P 500 is up.

BigPointValue

Dollar amount of 1 full point move.

Usage: `Value1 = BigPointValue * Close;`

Black

Specifies color Black (numeric value = 1) for plots and backgrounds.

BlockNumber

Returns the unique Security Block number attached to this computer.

Blue

Specifies the color Blue (numeric value = 2) for plots and backgrounds.

Usage: `Plot1(Value1, "Test", Blue);`

Book_Val_Per_Share

Returns calculated book value per share (common shares / outstanding shares).

BOOL

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

BoxSize

Refers to minimum price change needed to add an X or O to a Point & Figure chart.

BreakEvenStopFloor

Reserved for backward compatibility with previous versions of the product. Replaced by the reserved word **SetBreakEven**.

Buy

Initiates a long position. Covers any short positions & reverses an existing position.

Syntax: **Buy** [("Order Name")] [*num* of **shares**] execution instruction;
execution instructions: **this bar** on **close**, **next bar** at **market**,
next bar at *price* **stop**, **next bar** at *price* **limit**

Usage:
```
Buy Next Bar at Market;
Buy("Buy Close") 20 Shares This Bar on Close;
Buy 5 Contracts Next Bar at High + Range Stop;
Buy("BuyLimit") Next Bar at Price Limit;
```

BuyToCover

A trading strategy order to partially or completely cover short positions.

Syntax: **BuyToCover** [from entry ("MyTrade")] [*num* of **shares**] execution instruction;
execution instructions: **this bar** on **close**, **next bar** at **market**,
next bar at *price* **stop**, **next bar** at *price* **limit**

Usage:
```
BuyToCover Next Bar at Market;
BuyToCover From Entry ("BuyClose") Next Bar at 75 Stop
BuyToCover 5 Contracts Next Bar at Low + Range Stop;
BuyToCover From Entry ("BuyLimit") Next Bar at Price Limit;
```

By

Skip word ignored during execution.

Usage:
```
Value1 = (High-Close) / by 2;
```

BYTE

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

C

Abbreviation for Close. Returns the closing price of a referenced bar.

Usage:
```
If Price > Close of 1 Bar Ago Then Buy on Close;
```

Cancel

Used in conjunction with Alert to cancel a previously triggered alert.

Usage:
```
If {Any Condition} Then Cancel Alert;
```

Category

Category of symbol: 0=Future, 1=Future Option, 2=Stock, 3=Stock Option, etc.

Usage:
```
Value1 = Category          {returns a value of 3 for MSQ option of MSFT}
```

Ceiling

Returns the lowest integer greater than num.

Syntax: **Ceiling**(*Num*);
Num: a numeric value or expression

Usage:
```
Value1 = Ceiling(4.5)          {returns a value of 5}
```

CHAR

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

CheckAlert

Returns *True* for the last bar when **Enable Alert** check box is selected.

Usage:
```
If CheckAlert Then {Any Operation};
```

CheckCommentary

Returns *True* when the Analysis Commentary Tool is applied to the current bar.

Usage:
```
If CheckCommentary Then {Any Operation};
```

ClearDebug

Clears the contents of the Print Log tab of the EasyLanguage Output Bar.

Close

Returns the closing price of the bar being referenced.

Usage:
```
Value1 = Close of 1 Bar Ago ;
If Close > Close[1] Then Plot1(High,"ClosedUp");
```

Commentary

Sends EasyLanguage expression(s) to the Analysis Commentary window.

Usage:
```
Commentary("This is analysis commentary");
```

CommentaryCL

Sends EasyLanguage expression(s) to Analysis Commentary with a carriage return.

Usage:
```
CommentaryCL("This is a single line of commentary");
```

CommentaryEnabled

Returns *True* on any bar when the Analysis Commentary window is open.

Commission

Returns the commission setting from the current strategy's **Costs** tab.

CommodityNumber

Unique number representing a particular symbol in the Symbol Dictionary (optional).

Usage:
```
If CommodityNumber = 149 Then {Any Operation};
```

Contract

Specifies the number of units (contracts/shares) to trade within a trading strategy.

Usage:
```
Sell 1 Contract Next Bar at Market;
```

Contracts

Specifies the number of units (contracts/shares) to trade within a trading strategy.

Same as **Contract**.

Cosine

Returns the cosine value of *num* degrees.

Usage:
```
Value1 = Cosine(72);          {returns 0.3090 when num is 72 degrees}
```

Cost

Returns the value of the cost of establishing a leg or position.

Usage:
```
Plot1(Cost of Leg(1), "Cost");
```

Cotangent

Returns the cotangent value of *num* degrees.

Usage:
```
Value1 = Cotangent(45);       {returns 1.0 when num is 45 degrees}
```

Cross

Used to detect when values have crossed over/under or above/below another value.

Usage:
```
If Plot1 does Cross Above Plot2 Then {Any Operation};
```

Crosses

Used to detect when values have crossed over/under or above/below another value.

Usage:
```
If Value1 Crosses Below Value2 Then {Any Operation};
```

Current

Reserved for future use.

Current_Ratio

Returns the current ratio of a stock (Total Current Assets / Total Current Liabilities).

CurrentBar

Returns the number of the bar currently being evaluated.

CurrentContracts

The number of contracts in the current position (+*value* is long, -*value* is short).

CurrentDate

Returns the current date in the format YYMMDD or YYYMMDD.

Usage:
```
Value1 = CurrentDate;         {returns a value of 1011220 on December 20, 2001}
```

CurrentEntries

Number of entries currently open within a position.

Usage:
```
Value2 = CurrentEntries
```

CurrentTime

Returns the current time as HHMM using a 24-hour format.

Usage:
```
Value2 = CurrentTime          {returns a value of 1718 at 5:18 pm}
```

CustomerID

Returns the User ID number of the person to whom the software is registered.

Cyan

Specifies color Cyan (numeric value = 3) for plots and backgrounds.

D

Returns the closing date of the bar referenced. (Abbreviation for **Date**).

DailyLimit

Number of stocks/contracts allowed traded in 1 day.

DarkBlue

Specifies color Dark Blue (numeric value = 9) for plots and backgrounds.

DarkBrown

Specifies color Dark Brown (numeric value = 14) for plots and backgrounds.

DarkCyan

Specifies color Dark Cyan (numeric value = 10) for plots and backgrounds.

DarkGray

Specifies color Dark Gray (numeric value = 15) for plots and backgrounds.

DarkGreen

Specifies color Dark Green (numeric value = 11) for plots and backgrounds.

DarkMagenta

Specifies color Dark Magenta (numeric value = 12) for plots and backgrounds.

DarkRed

Specifies color Dark Red (numeric value = 13) for plots and backgrounds.

Data*N*

Used to reference information from a specified data stream.

Usage: `Value1 = Low of Data10` {returns the Low for the current bar from data stream 10}

DataCompression

The compression setting of the price data for the applied analysis technique.

Returns: 0 forTick, 1 for Intraday, 2 for Daily, 3 for Weekly, 4 for Monthly, 5 for Point & Figure.

Usage: `If DataCompression=2 Then {Any Operation}` {tests for daily bars}

DataInUnion

Reserved for future use.

Date

Returns the closing date of the bar referenced in YYYMMDD format.

Usage: `If Date < 990101 Then Buy This Bar on Close;`

DateToJulian

Converts calendar date to Julian date.

Syntax: **DateToJulian**(c*Date*);
cDate: numeric expression for the date in YYMMDD or YYYMMDD format.

Usage: `Value2 = DateToJulian(991024)` {returns Julian value of 36457}

Day

Reserved for backward compatibility. Replaced by **Bar**.

DayOfMonth

Returns the day of month (DD) portion of the specified calendar date.

Syntax: **DayOfMonth**(c*Date*);
cDate: numeric expression for the date in YYMMDD or YYYMMDD format.

Usage: `Value1 = DayOfMonth(991004)` {returns day value of 4}

DayOfWeek

Returns the day of week (0 for Sun., 1 for Mon., ..., 6 for Sat.) for a calendar date.

Syntax: **DayOfWeek**(c*Date*);
cDate: numeric expression for the date in YYMMDD or YYYMMDD format

Usage: `Value1 = DayOfWeek(1011024)` {returns 3 because Oct 24, 2001 is a Wednesday}

Days

Reserved for backward compatibility. Replaced by **Bars**.

Default

Used in plot statements to set a style to its default value.

Usage:
```
Plot1(Value1, "Plot1", Default, Default, 5);
```

DefineCustField

Reserved for future use.

DEFINEDLLFUNC

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

DeliveryMonth

Used for contracts that expire. Returns the month of expiration (1...12).

DeliveryYear

Used for contracts that expire. Returns the 3-digit year of expiration.

Description

Returns a string containing the description of the symbol if it is available.

Usage: `TextString= Description;` {symbol decription - blank if none available}

Dividend

Returns the Dividend paid any number of periods ago.

Usage: `Value1 = Dividend(2);` {the last dividend amount paid 2 periods ago}

Dividend_Yield

Most recent cash dividend paid (or declared) times the dividend payment frequency.

DividendCount

The number of times that dividends have been reported in the time frame considered.

DividendDate

Date of reported stock dividend any number of periods ago.

Syntax: **DividendDate**(*num*);
Num: number of periods ago, use 0 or no parameter for current period

Usage:
```
Value2 = DividendDate(4);  {the date of the dividend 4 periods ago}
```

DividendTime

The time at which a stock dividend was paid out any number of periods ago.

Syntax: **DividendTime**(*num*);
Num: number of periods ago, use 0 or no parameter for current period

Usage:
```
Value1 = DividendTime;      {the time of the last reported dividend}
```

Does

Skip word ignored during execution.

Usage:
```
If Plot1 Does Cross Over Plot2 Then {Any Operation}
```

DOUBLE

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

DownTicks

Number of ticks on a bar whose value is lower than the tick immediately preceding it (or an unchanged tick that follows a downtick).

DownTo

Instructs a loop's counter to decrement and exit the loop at a specified value.

Usage:
```
For Value5 = Length DownTo 0 Begin
    {Any Operations}
End;
```

DWORD

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

EasyLanguageVersion

Returns the EasyLanguage version currently installed (i.e., EL 2000i is version 5.1).

Usage:
```
If EasyLanguageVersion >= 5.0 Then {Any Operation}
```

EL_DateStr

Returns an 8 character YYYYMMDD string based on month, day, and year values.

Syntax: **EL_DateStr**(Month,Day,Year);
(Month) is a numeric expression representing a month (e.g., January = 01).
(Day) is a numeric expression representing the day of the month.
(Year) is a numeric expression representing a four-digit year.

Usage:
```
Value1 = EL_DateStr(09,05,1999){returns the string 19990905)
```

Else

Used to execute instructions when the specified 'If' condition returns *False*.

Usage:
```
If Condition1 Then
    {Operation done if condition is true}
Else
  {Operations done if condition is false} ;
```

End

Used with **Begin** to execute multiple statements based on a condition. See **Begin**.

Entry

An optional Exit parameter used to reference a specific, named entry.

Usage:
```
Sell from Entry ("MyTrade") Next Bar at Market;
```

EntryDate

Returns the entry date for the specified period in the format YYYMMDD.

Usage:
```
Value1 = EntryDate(2)        {the date of the entry 2 periods ago}
```

EntryPrice

Returns the entry price for the specified period.

Usage:
```
Value2 = EntryPrice(1)       {the price of the entry 1 period ago}
```

EntryTime

Returns the entry time for the specified period in the 24-hour format HHMM.

Usage:
```
Value1 = EntryTime(3)        {the time of the entry 3 periods ago}
```

EPS

Returns the reported earnings-per-share value for the specified period.

Usage:
```
Value2 = EPS(5)              {the Earnings-Per-Share 5 periods ago}
```

EPS_PChng_Y_Ago

The percent change in EPS this quarter vs. same quarter 1 year ago.

EPS_PChng_YTD

The percent change in EPS YTD earnings vs. YTD earnings same period 1 year ago.

EPSCount

The number of times that Earnings Per Share has been reported for a specified period.

EPSDate

The date on which Earnings Per Share were reported for the specified period.

Usage: `Value1 = EPSDate(2)` {the Earnings Per Share date 2 periods ago}

EPSTime

The time at which Earnings Per Share were reported for the specified period.

Usage: `Value1 = EPSTime` {the Earnings Per Share time for this period}

ExitDate

Returns the exit date for the specified position in the format YYYMMDD.

Usage: `Value2 = ExitDate(4)` {the exit date 4 positions ago}

ExitPrice

Returns the exit price for the specified position

Usage: `Value1 = ExitPrice(2)` {the exit price 2 positions ago}

ExitTime

Returns the exit time for the specified position in 24-hour HHMM format.

Usage: `Value1 = ExitTime(1)` {the exit time 1 position ago}

ExpValue

Returns the exponential value of the specified number.

Usage: `Value2 = ExpValue(4.5)` {returns a value of 90.0171}

False

Represents the logical value *False* when evaluating an expression or setting an input.

Usage: `Input:MyValue(False);` {initializes MyValue to False}

File

Sends information to a specified file from a print statement.

Syntax: **File**(*strFilename*);
strFileName: name of file to receive 'print' output

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

FileAppend

Appends a text string to the end of a specified file.

Syntax: **FileAppend**(*strFilename,strText*);
strFileName: name of file to which text will be appended
strText: text string containing information that will be added to the specified text file

Usage:
```
FileAppend("d:\myfile.txt","Add this text to the file");
```

FileDelete

Deletes the specified file.

Syntax: **FileDelete**(*strFilename*);
strFileName: path name of file

Usage:
```
FileDelete("e:\path\anyfile.txt");
```

FirstNoticeDate

Returns the first notice date of a futures contract, in YYYMMDD format.

FLOAT

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

Floor

Returns the highest integer less than the specified number.

```
Usage:  Floor(6.5)                    {returns a value of 6}
```

For

Executes a block of instructions a specified number of times within a loop.

```
Usage:  For N = 1 To 10 Begin
            Total = Total + Price[N];
        End ;                         {adds Price(N) to Total 10 times}
```

FracPortion

Returns the fractional portion of a number while retaining the sign.

```
Usage:  FracPortion(-1.72)            {returns a value of -0.72}
```

FreeCshFlwPerShare

Calculates and returns the Free Cash Flow Per Share value.

Friday

Specifies day of the week Friday (numeric value = 5).

From

Used with **Entry** to specify the name of a Long or Short entry in an Exit statement.

```
Usage:  Sell From Entry("MyTrade") Next Bar at 75 Stop ;
```

G_Rate_EPS_NY

The number of years over which the Earnings Per Share Growth Rate is calculated.

G_Rate_Nt_In_NY

The number of years over which the Net Income Growth Rate is calculated.

G_Rate_P_Net_Inc

The Net Income Growth Rate percentage for a stock.

GetBackgroundColor

Returns the current chart background color (see Appendix B for color values).

```
Usage:  Value1 = GetBackgroundColor;
```

GetCDRomDrive

Returns the drive letter of first CD-ROM found.

```
Usage:  Variable: Drive("D");
        Drive = GetCDRomDrive;
```

GetExchangeName

Returns the name of the Exchange for a symbol.

Usage: `Value1 = GetExchangeName;` {i.e. ,'NYSE' for the New York Stock Exchange}

GetPlotBGColor

Returns the background color of a cell on a grid.

Syntax: **GetPlotBGColor**(*PlotNum*);
PlotNum: value or expression representing the plot number

Usage: `Value2 = GetPlotBGColor(1);`

GetPlotColor

Returns the numeric color value of a chart's plot line or grid's foreground color.

Syntax: **GetPlotColor**(*PlotNum*);
PlotNum: value or expression representing the plot number

Usage: `Value1 = GetPlotColor(2);`

GetPlotWidth

Returns the width value of a plot line in a chart.

Syntax: **GetPlotWidth***PlotNum*);
PlotNum: value or expression representing the plot number

Usage: `Value2 = GetPlotWidth(1);`

GetStrategyName

Returns the strategy name as a string value.

GetSymbolName

Returns a string with the symbol name to which the analysis technique is applied.

GetSystemName

Reserved for backward compatibility. See **GetStrategyName**.

Gr_Rate_P_EPS

Returns the Earnings Per Share Growth Rate for a stock.

Green

Specifies color Green (numeric value = 4) for plots and backgrounds.

GrossLoss

Cumulative dollar total of all closed-out losing trades.

Usage: `Value1 = GrossLoss;` {returns -1000 for three losing trades of -500,-200, and -300}

GrossProfit

Cumulative dollar total of all closed-out winning trades.

Usage: `Value2 = GrossProfit;` {returns 800 for three winning trades of 100, 300, and 400}

H

Returns the highest price of the bar referenced. (abbreviation for **High**)

```
Usage:  Value1 = H[2];                    {returns the High of 2 bars ago}
```

High

Returns the highest price of the bar referenced.

```
Usage:  Value2 = High of 1 bar ago;       {returns the High of the previous bar}
```

Higher

Synonym for stop or limit orders depending on the context used within a strategy.

```
Usage1: Buy Next Bar at MyEntryPrice or Higher;          {Buy... Stop}
        BuyToCover Next Bar at MyExitPrice or Higher;    {BuyToCover... Stop}
Usage:2 SellShort Next Bar at MyEntryPrice or Higher;    {SellShort... Limit}
        Sell Next Bar at MyEntryPrice or Higher;         {Sell...Limit}
```

HistFundExists

True if historical fundamental info (EPS, Dividends, and Splits) exists for symbol.

I

Number of contracts outstanding at the close of a bar (abbreviation for **OpenInt**).

```
Usage:  Value1 = I of 1 bar ago;          {returns the open interest of the previous bar}
```

I_AvgEntryPrice

Returns the average entry price of each open entry in a pyramided position. For use when writing indicators and studies.

```
Usage:  Value2 = I_AvgEntryPrice;         {returns150 for opens entries at 130, 145, and 175)
```

I_ClosedEquity

Returns the profit or loss realized when a position is closed. For use when writing indicators and studies.

I_CurrentContracts

Returns the number of contracts held in all open entries. For use when writing indicators and studies.

```
Usage:  Value2= I_CurrentContracts;       {returns 3 for 3 open enries of 1 contract each)
```

I_MarketPosition

A strategy's current market position: 1 = long, -1 = short, 0 = flat. For use when writing indicators and studies.

```
Usage:  Value1 = I_MarketPosition;        {returns 1 if currently held position is Long)
```

I_OpenEquity

Returns the current gain or loss while a position is open.

If

Specifies condition(s) that must be met to execute a set of instructions.

Usage:
```
If Condition1 Then Begin
    {Operations done if condition is true}
End ;
```

IncludeStrategy

Used to include one strategy's EasyLanguage instructions in another.

Syntax: **IncludeStrategy**: "StrategyName",[Input1[,InputN...]];
StrategyName: Name of the strategy to be included
Input1: refers to one of the included strategy's inputs
InputN: additional input names separated by commas

Usage:
```
IncludeStrategy:"LowEntry",Price,BarCount;
```

IncludeSystem

Reserved for backward compatibility. Replaced by IncludeStrategy.

InitialMargin

Returns the Initial Margin Requirement of a position.

Usage:
```
If InitialMargin of Position > 500 Then {Any Operation}
```

Input

Used to declare an input name that accepts a user value when applying a technique.

Usage:
```
Input: Length(10);   {declares input 'Length' with an initial value of 10}
```

Inputs

Declares multiple inputs separated by commas. See Input.

Usage:
```
Inputs: Price(5.25), Length(8), Status(True);
```

Inst_Percent_Held

The percent of common stock held by institutions relative to total outstanding shares.

InStr

Returns the location of String2 within String1.

Syntax: **InStr**(String1,String2);
String1: Text string to be searched
String2: Word or phrase to be found in String1
Returns: Character position of the start of String2, if found. Zero if not found.

Usage:
```
Value1 = InStr("Net Profit Margin", "Profit");   {returns a 5}
```

INT

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

IntPortion

Returns the integer portion of the specified decimal number.

Syntax: **IntPortion**(*Num*);
Num: A numeric value or expression

Usage:
```
Value1 = IntPortion(4.125);          {returns a 4}
```

Is

Skip word ignored during execution.

Usage: **If** a **Close** is > 100 **Then** *{any operation} ;*

JulianToDate

Returns the calendar date YYYMMDD for the specified Julian date.

Syntax: **JulianToDate**(*jDate*);
jDate: numeric expression for the date in Julian format.

Usage:
```
Value2 = JulianToDate(36457);        {returns Date value of 991024}
```

L

Returns the lowest price of the bar referenced. (abbreviation for **Low**)

Usage:
```
Value1 = L[4];          {returns the Low of 4 bars ago}
```

LargestLosTrade

Returns the dollar value of the largest closed-out losing trade.

LargestWinTrade

Returns the dollar value of the largest closed-out winning trade.

Last_Split_Date

Returns the Date on which the last stock split was reported.

Last_Split_Fact

Returns the size or ratio of last stock split.

LastCalcJDate

Returns the Julian date of last completed bar.

LastCalcMMTime

Returns the time of last completed bar, in minutes since midnight.

Usage:
```
Value1 = LastCalcMMTime;     {returns a value of 540 if last bar was at 9:00 am}
```

LastTradingDate

Refers to the last day an option, future, position leg, or asset may be traded.

LeftSide

Used with ActivityBars to refer to actions on the left side of a bar.

Usage:
```
Value2 = GetCellChar(Close, Leftside, 3);
```

LeftStr

Returns the leftmost (starting) portion of a text string.

Syntax: **LeftStr**(*String*,*Length*);
String: A text string to evaluate. Must be enclosed in quotation marks.
Length: The number of characters to return from the start of *String*.

Usage:
```
Value1 = LeftStr("Net Profit", 3);        {returns the word "Net"}
```

LightGray

Specifies color Light Gray (numeric value = 16) for plots and backgrounds.

Limit

In an entry or exit order, means 'or higher' or 'or lower', depending on the context.

Usage:
```
Buy Next Bar at 75 Limit; {enters a long position at a price of 75 or lower}
SellShort Next Bar at 75 Limit; {enters a short position at 75 or higher}
```

Log

Returns the natural logarithm of a number.

Usage:
```
Value1 = Log(172);                        {returns a log value of 5.1475}
```

LONG

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

Low

Returns the lowest price of the bar referenced.

Usage:
```
Value2 = Low of 1 bar ago;                {returns the Low of the previous bar}
```

Lower

Synonym for stop or limit orders depending on the context used within a strategy.

Usage1:
```
Buy Next Bar at MyEntryPrice or Lower;          {Buy... Limit}
BuyToCover Next Bar at MyExitPrice or Lower;{BuyToCover... Limit}
```

Usage2:
```
SellShort Next Bar at MyEntryPrice or Lower;{SellShort... Stop}
Sell Next Bar at MyEntryPrice or Lower;         {Sell...Stop}
```

LowerStr

Used to convert a string expression to lowercase letters.

Usage1:
```
Value1 = LowerStr("My TextString") ;          {returns "my textstring"}
```

LPBOOL

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

LPBYTE

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

LPDOUBLE

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

LPDWORD

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

LPFLOAT

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

LPINT

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

LPLONG

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

LPSTR

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

LPWORD

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

Magenta

Specifies color Magenta (numeric value = 5) for plots and backgrounds.

MakeNewMovieRef

Creates new movie reference number.

Usage: `Print(MakeNewMovieRef = 1);`

Margin

Returns the margin setting from the **Trade costs** section of the strategy's **General** tab

Market

Order type referring to the opening price of the next bar.

Usage: `Buy Next Bar at Market;`

MarketPosition

The market position (1 = long, -1 = short, 0 = flat) of the specified position.

Syntax: **MarketPosition**(*Num*)
Num: number of positions ago

Usage: Value1 = **MarketPosition**(2) ; {returns1 if long 2 posittions ago was long}

MaxBarsBack

The minimum number of bars required to evaluate a study or trading strategy.

MaxBarsForward

Represents the number of bars to the right of the last bar on the chart.

MaxConsecLosers

Represents the longest chain of consecutive closed-out losing trades.

MaxConsecWinners

Represents the longest chain of consecutive closed-out winning trades.

MaxContracts

The maximum number of contracts held during the specified position.

Syntax: **MaxContracts**(*Num*)
Num: number of positions ago.

Usage: Value1 = **MaxContracts**(2) ; {returns number of contracts held 2 posittions ago}

MaxContractsHeld

Maximum number of contracts held at any one time.

MaxEntries

The maximum number of entry strategies for the specified position.

Syntax: **MaxEntries**(*Num*)
Num: number of positions ago.

MaxIDDrawDown

The largest drop in equity (in dollars) throughout the entire trading period.

MaxList

Returns the highest value of the listed inputs.

Syntax: **MaxList**(*Num1[,NumN...]*)
Num1 the first value or expression to compare
NumN additional values to compare separated by commas

Usage:
```
Value1 = MaxList(45, 72, 86, 125, 47);
```
{returns a value of 125}

MaxList2

Returns the second highest value of the listed inputs. See **MaxList** for syntax.

Usage:
```
Value2 = MaxList2(18, 67, 98, 24, 65, 19);
```
{returns a value of 67}

MaxPositionLoss

Dollar amount of largest loss for the specified position.

Syntax: **MaxPositionLoss**(*Num*)
Num: number of positions ago.

MaxPositionProfit

Dollar amount of largest gain for the specified position.

Syntax: **MaxPositionProfit**(*Num*)
Num: number of positions ago.

MessageLog

Reserved for backward compatibility.

MidStr

Returns the middle portion of a text string.

Syntax: **MidStr** (*String*,*Location*,*Size*) ;
String: text expression to evaluate
Location: starting character position of the text string to be returned
Size: length of the text string to be returned

Usage:
```
Value1 = MidStr("Net Profit Value", 5, 6) {returns the word 'Profit'}
```

MinList

Returns the lowest value of the listed inputs.

Syntax: **MinList**(*Num1[,NumN...]*)
Num1 the first value or expression to compare
NumN additional values to compare separated by commas

Usage:
```
Value1 = MinList(45, 72, 86, 125, 47);      {returns a value of 45}
```

MinList2

Returns the second lowest value of the listed inputs. See **MinList** for syntax.

Usage:
```
Value2 = MinList2(18, 67, 98, 24, 65, 19)   {returns a value of 19}
```

MinMove

Minimum tick movement of stock/future symbol.

Usage:
```
Value1 = MinMove * PriceScale      {returns the smallest price increment}
```

Moc

Reserved for future use.

Mod

Divides two numbers and returns the remainder.

Syntax: **Mod**(*Num,Divisor*)
Num: any value or expression
Divisor: any numeric expression representing the divisor.

Usage:
```
Value1 = Mod(17, 5);          {divides 17 by 5 and returns 2 as the remainder}
```

Monday

Specifies day of the week Monday (numeric value = 1).

MoneyMgtStopAmt

Reserved for backward compatibility with previous versions of the product. Replaced by the reserved word **SetStopLoss**.

Month

Returns the month (MM) portion of the specified calendar date, from 1 to 12.

Syntax: **Month**(c*Date*)*;*
cDate: numeric expression for the date in YYMMDD or YYYMMDD format.

Usage:
```
Value1 = Month(991004)                {returns month value of 10}
```

MULTIPLE

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

Neg

Returns the absolute negative value of a number.

```
Usage:  Value1 = Neg(17);                    {returns a value of -17}
        Value2 = Neg(-9);                    {returns a value of -9}
```

Net_Profit_Margin

Calculates and returns the Net Profit Margin (Income after Taxes / Total Revenue).

NetProfit

Cumulative dollar total of all closed-out trades, both winning and losing.

```
Usage:  Value1 = NetProfit {returns 1000 for three closed trades of -500, 1200 and 300}
```

NewLine

Adds carriage return/linefeed in FileAppend and commentary/file output strings.

```
Usage:  FileAppend("c:\my.txt", "Text Line1" + NewLine + "Line2");
```

Next

Used in conjunction with **Bar** to reference the next bar in a trading strategy.

```
Usage:  Buy Next Bar at Market;
```

NoPlot

Removes a plot from the current bar in a chart or cell in a grid.

```
Usage:  If Close > Close[1] Then
              Plot1(High, "CloseUp")          {plots 'CloseUp' on the bar}
        Else
              NoPlot(1);                      {removes a previous plot from the bar}
```

Not

Reserved for future use.

NthMaxList

Returns the Nth highest value of the listed inputs.

Syntax: **NthMaxList**(*N*,*Num1[,NumN...]*)

N: an integer representing the rank in the list (1st, 2nd, 3rd, etc.)
Num1: the first value or expression to compare
NumN: additional values to compare separated by commas

```
Usage:  Value1 = NthMaxList(2, 45, 72, 86, 125, 47);  {returns a value of 86}
```

NthMinList

Returns the Nth lowest value of the listed inputs. See syntax as **NthMaxList**.

```
Usage:  Value1 = NthMinList(2, 45, 72, 86, 125, 47);  {returns a value of 47}
```

Numeric

Defines an input that expects a number passed by value.

Usage: `Input: Price(Numeric);` {accepts a numeric value for Price}

NumericArray

Defines an input that expects a number passed by value for each array element.

Usage: `Input: MyArray[n](NumericArray)` {accepts numeric inputs by value}

NumericArrayRef

Defines an input that expects a numeric variable passed by reference for each array element.

Usage: `Input: MyArray[n](NumericArrayRef)` {accepts numeric inputs by reference}

NumericRef

Defines an input that expects a numeric variable passed by reference.

Usage: `Input: Price(NumericRef);` {accepts a numeric variable reference for Price}

NumericSeries

Defines an input as a numeric series expression with price history.

Usage: `Input: Price(NumericSeries);` {a numeric input allowing previous bar history}

NumericSimple

Defines an input as a numeric simple expression.

Usage: `Input: Price(NumericSimple);` {a numeric input not allowing bar history}

NumFutures

Returns the total number of futures contracts associated with a future symbol root.

Usage: `Value1 = NumFutures of Asset;`

NumLosTrades

Returns the total count of closed-out losing trades.

NumToStr

Converts the specified numeric expression to a string expression.

Syntax: **NumToStr**(*Num*,*Dec*);
Num: a numeric expression to be converted to a string
Dec: the number of decimal places for the string version of the value

Usage: `Value1 = NumToStr(1170.5, 2) ;` {returns the text string '1170.50'}

NumWinTrades

Total count of closed-out winning trades.

O

Abbreviation for Open. Returns the opening price of a referenced bar.

Usage: `If Price < O of 1 Bar Ago Then SellShort at Market;`

Of

Skip word ignored during execution.

Usage:
```
If Close of Data1 = Highest(High, 14) Then {any operation} ;
```

On

Skip word ignored during execution.

Usage:
```
Buy 100 Contracts on Next Bar Open;
```

Open

Returns the opening price of the bar referenced.

Usage:
```
Value1 = Open of 2 Bars Ago;
```

OpenInt

The open interest, or number of contracts outstanding, at the close of a specific bar.

Usage:
```
Value2 = Average(OpenInt, 10);    {returns the average OpenInt over 10 bars}
```

OpenPositionProfit

Returns the gain or loss of current open position (only used with strategies).

Or

Links 2 true/false expressions together. *True* if either expression is true.

Usage:
```
If Plot1 Crosses Above Plot2 Or Plot2 > 5 Then Begin
  {any operations}                    {done if either condition is true}
End;
```

Over

Used only with **Crosses** to detect a value crossing over, or above, another value.

Usage:
```
If Plot1 Crosses Over Plot2 Then {Any Operation} ;
```

Pager_DefaultName

Returns the string containing of the default Message Recipient as specified in the **Messaging** tab under the **File - Desktop Options** menu.

Usage:
```
Name = Pager_DefaultName;
Pager_Send(Name, "Buy 200 AMD at Market");
```

Pager_Send

Sends a text message to a specified pager recipient (if pager module enabled).

Syntax: **Pager_Send**(*sTo*,*sMessage*);

sTo: text string containing the name of the message recipient

sMessage: text string containing the message contents

Usage:
```
Pager_Send("Joe Trader", "Buy 200 AMD at Market");
```

PercentProfit

Percentage of all closed-out winning trades.

Usage: Value1 = **PercentProfit**; {returns 80 if 8 of 10 trades were winners}

Place

Retained for backward compatibility. Skip word.

PlayMovieChain

Queues and plays the movie chain with the specified reference number.

Usage: `Condition1 = PlayMovieChain(1);` {plays the movie chain with ref number 1}

PlaySound

Plays the specified sound file (.wav file).

Usage: `Condition1 = PlaySound("c:\sounds\thatsabuy.wav");`

Plot

References the value of a specified plot.

Syntax: **Plot**(*n*);
n: plot number ranging from 1-4

Usage: `If Plot(Value1) < Close Then Buy Next Bar on Open;`

Plot1

Displays an expression (numeric or text) in a price chart or grid.

Syntax: **Plot1**(*Value[,sName[,fgColor,[bgColor[,Width]]]]*);
Value: a numeric or text string expression or value to display on a chart or grid
sName: text string containing the name of the plot (optional)
fgColor: color number (or Default) of the plotted object or text (optional)
bgColor: color number (or Default) of the cell background in a grid (optional, ignored for charts)
Width: the thickness of a line to be plotted on a chart (optional, ignored for grids)

Usage:
```
Plot1(Value) ;
```
or
```
Plot1(Value, "My Plot Name", Red, Default, 0) ;
```

Plot2

Displays an expression in a price chart or grid. See **Plot1** for syntax and usage.

Plot3

Displays an expression in a price chart or grid. See **Plot1** for syntax and usage.

Plot4

Displays an expression in a price chart or grid. See **Plot1** for syntax and usage.

PlotPaintBar

For use with PaintBar studies, enables you to paint the entire bar, or part of the bar, with a single instruction.

Syntax: **PlotPaintBar**(*High, Low[, Open, Close[, "PlotName"[,fgColor, [bgColor, Width]]]]);*
High: the upper price limit to paint
Low: the lower price limit to paint
Open: (optional) paints the opening tick mark
Close: (optional) paints the closing tick mark
PlotName: (optional) name used when referencing the plot
fgColor: (optional) color number (or Default) of the paint color
bgColor: (optional) color number (or Default) of the background (currently ignored with charts)
Width: (optional) the thickness of the lines to be plotted

Usage1: `PlotPaintBar(High,Low,Open,Close,"My Plot Name");`
or
Usage2: `PlotPaintBar(High,Low);`

PlotPB

Abbreviated version of **PlotPaintBar** (see PlotPaintBar).

PM_GetCellValue

Returns the intensity value of a cell at the specified column and price location.

Syntax: **PM_GetCellValue**(*ColNum,Price*);
ColNum: the ProbabilityMap column number where the cell is located
Price: the price location of the cell

Usage: `Value1 = PM_GetCellValue(12,High) ;`

PM_GetNumColumns

Returns the number of columns in a ProbabilityMap array.

Usage: `Value1 = PM_GetNumColumns;`

PM_GetRowHeight

Returns the height or increment of the rows in a ProbabilityMap study.

PM_High

Returns the value of the upper range of a ProbabilityMap grid.

PM_Low

Returns the value of the lower range of a ProbabilityMap grid.

PM_SetCellValue

Sets the location and intensity of ProbabilityMap cells.

Syntax: **PM_SetCellValue**(*ColNum,Price,Value*) ;
ColNum: the ProbabilityMap column of the cell to be set
Price: the price location of the cell within the column
Value: a numeric expression representing the intensity of the cell

Usage: `PM_SetCellValue(5,80,10) ;` {sets intensity of cell to 10}

PM_SetHigh

Sets the upper range value of a ProbabilityMap.

Syntax: **PM_SetHigh**(*Price*)
Price: a numeric expression or value for a price

Usage:
```
PM_SetHigh(Highest(High,50)); {sets the PM top to the Highest High}
```

PM_SetLow

Sets the lower range value of a ProbabilityMap.

Syntax: **PM_SetLow**(*Price*)
Price: a numeric expression or value for a price

Usage:
```
PM_SetLow(14.5) ; {sets the PM bottom to a price of 14.50}
```

PM_SetNumColumns

Sets the number of columns in a probability map array.

Syntax: **PM_SetNumColumns**(*Num*);
Num: a numeric expression or value representing the desired number of columns

Usage:
```
PM_SetNumColumns(PM_BarColumns) ;
```

PM_SetRowHeight

Sets the height of the rows for a ProbabilityMap grid.

Syntax: **PM_SetRowHeight**(*RowHeight);*
RowHeight: a numeric expression or value for the height of each PB row

Usage:
```
PM_SetRowHeight(.125) ; {sets the PM row height to a price of .125}
```

Pob

Retained for backward compatibility. Replaced by **Limit**.

Point

The minimal fractional value a symbol can move (one increment in the Price Scale).

Usage:
```
Sell This Bar at EntryPrice - 1 Point Stop;
```

POINTER

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

Points

Represents multiple 'Point' increments of the Price Scale. See **Point**.

Usage:
```
Buy This Bar at Close - 3 Points Stop;
```

PointValue

The dollar value per share of one increment on the price scale. Calculated as Big Point Value divided by the Price Scale using the values specified in the symbol dictionary.

Usage:
```
Value1 = PointValue; {returns 2.5 for S&P Futures}
```

Pos

Returns the absolute positive value of a number.

```
Usage:  Value1 = Pos(17);          {returns a value of 17}
        Value2 = Pos(-9);          {returns a value of 9}
```

PositionProfit

Returns the current gain (positive) or loss (negative) of the specified position.

```
Usage:  Value1 = PositionProfit;   {returns -1.00 if the position had a loss of 1.00}
```

Power

Returns the number raised to the specified power.

Syntax: **Power**(*Num,Exponent*);
Num: a numeric expression or value
Exponent: the power by which to raise the number

```
Usage:  Value1 = Pow(2,3);         {returns 8 based on 2^3}
```

Price_To_Book

Stock price vs. net worth of stock company.

PriceScale

Price scale of stock/future symbol (inverted).

```
Usage:  Value2 = PriceScale        {returns 100 for the S&P 500 Futures representing 1/100}
```

Print

Sends information to the Output Bar in the EasyLanguage PowerEditor or, if specified, to an alternate output location (a file or the default printer).

Syntax: **Print** (*Item1[,ItemN...]*) ;
Item1: a string or numeric expression
ItemN: additional strings or expressions separated by commas

```
Usage:  Print(Date, Time, Close);          {prints the 3 values to the Print Log}
        window}
Usage1: Print(Printer, D, T, C);           {prints the same 3 values to the default printer}
Usage2: Print("c:\myfile.txt", D, T, C);   {prints the 3 values to the specified file}
```

Printer

Sends information to the default printer from a Print statement.

```
Usage:  Print(Printer,"Today is: ", Date);  {sends output to the default printer}
```

Product

Number representing the TradeStation Technologies application currently being used.

Product Name	Product Number
TradeStation	0

```
Usage:  If Product = 0 Then Plot1(Value1, "TS Indicator");
```

Profit

Reserved for future use.

ProfitTargetStop

Retained for backward compatibility with previous versions of the product. Replaced by the reserved word **SetProfitTarget**.

Protective

Reserved for future use.

Quick_Ratio

Calculated as (cash + short term investment + accounts receivable) / current liabilities.

Random

Returns a pseudo-random number between 0 and *num*.

Syntax: **Random**(*num*) ;
Num: value that determines the range of possible numbers, starting with 0 and ending with *Num*

Usage:
```
Value1 = Random(37);          {randomly returns any value between 0 and 37}
```

Red

Specifies color Red (numeric value = 6) for plots and backgrounds.

Repeat

Reserved for future use.

Ret_On_Avg_Equity

Calculated as (income available to common stockholders / average common equity).

RevSize

Reversal size of a Point & Figure chart. Set on the **Settings** tab under **Format Symbol**.

RightSide

Used with ActivityBars to refer to actions on the right side of a bar.

Usage:
```
AB_AddCell(Open, Rightside, "A", 7, 1);
```

RightStr

Returns the rightmost (ending) portion of a text string.

Syntax: **RightStr**(*String,Length*);
String: A text string to evaluate. Must be enclosed in quotation marks.
Length: The number of characters to return from the end of *String*.

Usage:
```
Value1 = RightStr("Net Profit", 6);      {returns the word "Profit"}
```

Round

Returns a number rounded to nearest precision.

Divides two numbers and returns the remainder.

Syntax: **Round**(*Num,Precision*)
Num: any value or expression
Precision: the number of decimal places to keep

Usage:
```
Value1 = Round(9.5687, 3);                {returns a value of 9.569}
```

Saturday

Specifies day of the week Saturday (numeric value = 6).

Screen

Reserved for future use.

Sell

A trading strategy order to partially or completely liquidate a long position.

Syntax: **Sell** [from entry ("MyTrade")] [*num* of **shares**] [execution instruction];
execution instructions: **this bar** on **close**, **next bar** at **market**,
next bar at *price* **stop**, **next bar** at *price* **limit**

Usage:
```
Sell Next Bar at Market;
Sell From Entry ("BuyClose") Next Bar at 75 Stop
Sell 5 Contracts Next Bar at Low + Range Stop;
Sell From Entry ("BuyLimit") Next Bar at Price Limit;
```

SellShort

Initiates a short position. Closes any open positions & reverses an existing position.

Syntax: **SellShort** [("Order Name")] [*num* of **shares**] [execution instruction];
execution instructions: **this bar** on **close**, **next bar** at **market**,
next bar at *price* **stop**, **next bar** at *price* **limit**

Usage:
```
SellShort Next Bar at Market;
SellShort("Buy Close") 20 Shares This Bar on Close;
SellShort 5 Contracts Next Bar at Low + Range Stop;
SellShort("BuyLimit") Next Bar at Price Limit;
```

Sess1EndTime

Ending time of the first trading session for the security in 24-hour format.

Usage:
```
Value2 = Sess1EndTime;          {returns 1615 for IBM trading on the NYSE}
```

Sess1FirstBarTime

Completion time of the first bar in the first session in 24-hour format.

Usage:
```
Value2 = Sess1FirstBarTime;     {returns 1000 for IBM using 30 min bars}
```

Sess1StartTime

Starting time of the first trading session for the security in 24-hour format.

Usage:
```
Value1 = Sess1StartTime;        {returns 0930 for IBM trading on the NYSE}
```

Sess2EndTime

Ending time of the second trading session for the security in 24-hour format.

Usage:
```
Value2 = Sess2EndTime;          {returns 0745 for US Treasury Bonds on CBOE}
```

Sess2FirstBarTime

Completion time of the first bar in the second session in 24-hour format.

Usage:
```
Value1 = Sess2FirstBarTime;     {returns 1715 for S&P 500 Futures on 30 min bars}
```

Sess2StartTime

Starting time of the second trading session for the security in 24-hour format.

```
Usage:  Value1 = Sess2StartTime;        {returns 1530 for US Treasury Bonds on CBOE}
```

Sessions

Returns a numeric expression representing the number of sessions.

SetBreakEven

Sets a breakeven stop; specifies the profit required before placing the stop. Used by the trading strategy **BreakEven StopFloor**.

```
Syntax:  SetBreakEven(Price)
         Price: the floor, or minimum equity, needed for the stop to become active
Usage:   SetStopPosition;                          {can also use SetStopContract}
         SetBreakEven(250);          {places a breakeven stop after a $250 position profit}
```

SetDollarTrailing

Sets a dollar risk trailing stop; specifies the maximum tolerated loss amount (in dollars) of the maximum open position profit. Used by the trading strategy **Dollar Risk Trailing**.

```
Syntax:  SetDollarTrailing(Amount)
         Amount: the dollar amount you are willing to risk per position or per contract/share
Usage:   SetStopPosition;                          {can also use SetStopContract}
         SetDollarTrailing(500); {sets dollar risk trailing stop at $500 for entire position}
```

SetExitOnClose

Sets a stop to exit the position on the last bar of the day (for intraday charts). Used by the trading strategy **Close at End of Day**.

```
Usage:  SetExitOnClose;                            {exits positions at end of day}
```

SetPercentTrailing

Sets a percent risk trailing stop; specifies the profit that must be reached to activate stop and the maximum tolerated loss amount (as a percentage) of the maximum open position profit. Used by the trading strategy **PercentRisk Trailing**.

```
Syntax:  SetPercentTrailing(Amount,Percent)
         Amount: the dollar amount representing the minimum needed to activate the stop
         Percent: the percentage of the maximum equity needed to be lost to close the trade
Usage:   SetStopPosition;                          {can also use SetStopContract}
         SetPercentTrailing(500,15);     {exits after returnof 15% over $500 earned}
```

SetPlotBGColor

Assigns a specified background color to grid cells for an indicator.

```
Syntax:  SetPlotBGColor(Num,Color)
         Num: plot number to set
         Color: EasyLanguage color word (e.g., red, black,white) or color number
Usage:   SetPlotBGColor(1, Green);     {sets background color of Plot1 cells to Green}
```

SetPlotColor

Sets the color value of a chart's plot line or grid's foreground text color.

Syntax: **SetPlotColor**(*Num,Color*);
Num: plot number to set
Color: EasyLanguage color word (e.g. red, black,white) or color number

Usage:
```
SetPlotColor(2, Blue);          {sets foreground color of Plot2 text to Blue}
```

SetPlotWidth

Modifies the width value (thickness) of a plot line in a chart.

Syntax: **SetPlotWidth**(*Num,Width*);
Num: plot number to set
Width: Numeric expression representing the plot's width

Usage:
```
SetPlotWidth(1, 5);                {sets the line width of Plot1 to 5}
```

SetProfitTarget

Sets a profit target stop; this reserved word specifies the profit required in order to exit the position. Used by the trading strategy **Profit Target**.

Syntax: **SetProfitTarget**(*Amount*)
Amount: the dollar value of the profit target

Usage:
```
SetStopContract;
SetProfitTarget(400);          {exits a position once it has returned $400}
```

SetStopContract

Instructs TradeStation to evaluate all stop values of a strategy on a per contract (entry) basis. Use **SetStopPosition** to evaluate stop values on a per position basis.

Usage:
```
SetStopContract;                {sets a stop for individual contract (entry)}
SetStopLoss(50) ;
```

SetStopLoss

Sets a stop loss order (money management stop); specifies the amount (in dollars) you are willing to lose on the position/contract before it is liquidated. Used by the trading strategy *Stop Loss*.

Syntax: **SetStopLoss**(*Amount*)
Amount: the dollar amount that must be incurred before position/contract is liquidated

Usage:
```
SetStopContract;                {can also use SetStopContract}
SetStopLoss(2);                 {exits long position when down $2 per contract}
```

SetStopPosition

Instructs TradeStation to evaluate all stop values of a strategy on a per position basis. To evaluate all stop values on a per contract (entry) basis, use **SetStopContract**.

Usage:
```
SetStopPosition;
SetStopLoss(1200);          {places a stop loss order of $1200 for entire position}
```

SGA_Exp_By_NetSales

Annualized growth rate percentage of sales (calculated from the total revenue divided by the number of outstanding shares).

Share

Used to specify a contract/share for a particular Buy, SellShort, or exit order.

Usage:
```
Buy 1 Share Next Bar at Market;
```

Shares

Used to specify the number of contracts/shares for a particular Buy, SellShort, or exit order.

Usage:
```
Sell 5 Shares Next Bar at Open;
```

Sign

Returns 1 for a positive num, -1 for a negative num, and 0 for a num of zero.

Syntax: **Sign**(*Num*)

Num: a numeric value or expression.

Usage:
```
Value1 = Sign(-9.5687)          {returns a value of -1}
```

Sine

Returns the sine value of *num* degrees.

Usage:
```
Value1 = Sine(72);          {returns 0.9511 when num is 72 degrees}
```

Skip

Reserved for future use.

Slippage

Returns the slippage per contract from **Trade costs** section of the strategy's **General** tab.

SnapFundExists

True if snapshot fundamental data exists in the data stream; False otherwise.

Spaces

Specifies the number of blank spaces to add to a text or commentary string.

Usage:
```
Print("Close" + Spaces(5) + NumToStr(Close, 3));
```

Square

Returns the square (2nd power) of the specified number.

Syntax: **Square**(*Num*)

Num: a numeric value or expression

Usage:
```
Value1 = Square(6.23)          {returns a value of 38.8219}
```

SquareRoot

Returns the square root of the specified number.

Syntax: **SquareRoot**(*Num*)
Num: a numeric value or expression

Usage:
```
Value1 = SquareRoot(5);          {returns a value of 2.2361}
```

StartDate

Reserved for future use.

StockSplit

Ratio of the stock split reported during a certain period.

Usage:
```
Value2 = StockSplit(2);          {returns the split ratio reported 2 periods ago}
```

StockSplitCount

The number of stock splits that have been reported in a given time frame.

StockSplitDate

The date on which a stock split was reported during a certain period.

Usage:
```
Value1 = StockSplitDate(3);      {date of a stock split reported 3 periods ago}
```

StockSplitTime

The time at which a stock split occurred during a certain period.

Usage:
```
Value2 = StockSplitTime;         {time of the last reported stock split}
```

Stop

In an entry or exit order, means 'or higher' or 'or lower', depending on the context.

Usage:
```
Buy Next Bar at 65 Stop;     {enters a long position at a price of 65 or higher}
Sell Next Bar at 65 Stop;    {exits a long position at a price of 65 or lower}
```

String

Defines a function input that accepts a string expression value.

Usage:
```
Input: MyMessage(String);        {accepts a text string value}
```

StringArray

Defines a function input array that accepts multiple string expressions.

Usage:
```
Input: Messages[n](StringArray);     {array that accepts text strings}
```

StringArrayRef

Defines a function input array that accepts multiple string references.

Usage:
```
Input: Note[n](StringArrayRef);  {array that accepts strings by reference}
```

StringRef

Defines a function input that accepts a string expression by reference.

Usage:
```
Input: SomeText(StringRef);      {accepts a text string by reference}
```

StringSeries

Defines a function input that accepts string expressions that include history.

Usage: `Input: SomeText(StringSeries);` {accepts text strings with history}

StringSimple

Defines a function input that accepts simple string expressions without history.

Usage: `Input: SomeText(StringSimple);` {accepts text strings without history}

StrLen

The number of characters that make up a text string.

Syntax: **StrLen**(*String*)

String: a text string expression (a variable or text contained within quote marks).

Usage: `Value1 = StrLen("Net Profit");` {returns a count of 10 characters}

StrToNum

Returns the numerical value of a text string, zero if the string not numeric.

Syntax: **StrToNum**(*String*)

String: a text string expression (a variable or text contained within quote marks).

Usage: `Value2 = StrToNum("1170.50");` {returns the numeric value 1117.5}

SumList

Returns the sum of all listed inputs.

Syntax: **SumList**(*Num1[,NumN...]*)

Num1: the first value or expression

NumN: additional values to add separated by commas

Usage: `Value1 = SumList(45, 72, 86, 125, 47);` {returns a value of 375}

Sunday

Specifies day of the week Sunday (numeric value = 0).

SymbolName

Returns a string expression representing the symbol name. See also **GetSymbolName**.

SymbolNumber

Number representing the GlobalServer symbol. See **CommodityNumber** and **Cusip**.

SymbolRoot

Returns a text string representing the root of the symbol (for futures and options only).

T

Abbreviation for Time. Returns the closing time of a referenced bar in 24-hour format.

Usage:
```
If T of 1 Bar Ago >= 1100 Then
    Buy at Market;
```
{buys at or after 11 AM}

Tangent

Returns the tangent value of *num* degrees.

Usage: `Value1 = Tangent(72);` {returns 3.0776 when *num* is 72 degrees}

Target

Reserved for future use.

TargetType

Reserved for future use.

Text

Retained for backward compatibility.

Text_Delete

Deletes the specified text object.

Syntax: **Text_Delete**(*TX_Ref*)
TX_Ref: a numeric expression representing the object identification number
Returns: status code indicating whether or not operation was successful

Usage1: `Text_Delete(3);` {deletes text object number 3}
Usage2: `Value1 = Text_Delete(2);` {returns status code after deleting text object 2}

Text_GetColor

Returns the color value of the specified text object. (see **Text_Delete** for syntax)

Usage: `Value1 = Text_GetColor(4);` {returns the color of text object 4}

Text_GetDate

Returns the date of the left edge of the specified text object.

Syntax: **Text_Delete**(*TX_Ref*)
TX_Ref: a numeric expression representing the object identification number

Usage: `Value1 = Text_GetDate(2);` {returns the date of text object 2}

Text_GetFirst

Returns the text object id number for the first object of a specified type.

Syntax: **Text_GetFirst**(*Type*)
Type: identifies the origin of the requested first text object
1 = text created by an analysis technique
2 = text created by the text drawing object only, and
3 = text created by either the text drawing object or an analysis technique
Returns: status code indicating whether or not operation is successful

Usage: `Value2 = Text_GetFirst(2);` {returns the id of first text drawing object}

Text_GetHStyle

Gets the horizontal placement style of the specified text object. (see **Text_Delete** for syntax)
Returns: 0 for left, 1 for right, 2 for center, or status code if operation not successful

Usage: `Value1 = Text_GetHStyle(5);` {returns horiz style of text object 5}

Text_GetNext

Returns the text object id for the next object of a specified type after specified object.

Syntax: **Text_GetNext**(*TX_Ref*, *Type*)

TX_Ref: a numeric expression representing the object identification number

Type: identifies the origin of the next text object

1 = text created by an analysis technique

2 = text created by the text drawing object only, and

3 = text created by either the text drawing object or an analysis technique

Returns: status code indicating whether or not operation is successful

Usage: *Value2* = ***Text_GetNext****(2,1);* {returns the id of analysis text after id 2}

Text_GetString

Returns the text string of the specified text object. (see **Text_Delete** for syntax)

Usage: `TextValue1 = Text_GetString(3);` {returns text string of object number 3}

Text_GetTime

Returns the time of the left edge of the specified text object. (see **Text_Delete** for syntax)

Usage: `Value2 = Text_GetTime(4);` {returns the time of text object 4}

Text_GetValue

Returns the price (vertical axis) of the specified text object. (see **Text_Delete** for syntax)

Usage: `Value1 = Text_GetValue(2);` {returns the price of text object 2}

Text_GetVStyle

Gets the vertical placement style of the specified text object. (see **Text_Delete** for syntax)

Returns: 0 for top, 1 for bottom, 2 for center, or error code if operation not successful

Usage: `Value2 = Text_GetVStyle(5);` {returns vert style of text object 5}

Text_New

Creates and draws a new text object at a specified date, time, and price location.

Syntax: **Text_New**(*cDate,Time,Price*,*Text*)

cDate: date in YYYMMDD format

Time: time in HHMM 24-hour format

Price: value or numeric expression of the price

Text: text variable or text expression within quotes

Returns: object number if successful, or error code if operation not successful

---

Note*: Drawing objects are numbered by type in the order they are created, from 0 to n. Therefore, 0 is the identification number of the first drawing object of that type created, and n is the last object of the same type created.*

Usage:
```
Value1 = Text_New(Date, Time, High + 1, "Stock Split");
```

Text_SetColor

Changes the color of the specified text object.

Syntax: **Text_SetColor**(*TX_ref, Color*)
TX_Ref: a numeric expression representing the object identification number
Color: the color name or numeric value

Usage:
```
Text_SetColor(3,red);                    {sets text object 3 to color red}
```

Text_SetLocation

Moves specified text object to a new date, time, and price location.

Syntax: **Text_SetLocation**(*TX_ref, cDate,Time,Price,Text*)
TX_Ref: a numeric expression representing the object identification number
cDate: date in YYYMMDD format
Time: time in HHMM 24-hour format
Price: value or numeric expression of the price
Text: text variable or text expression within quotes
Returns: 0 if successful, or error code if operation not successful

Usage:
```
Value1 = Text_SetLocation(2,990114,1500,24.5); {moves text obj 2}
```

Text_SetString

Changes the text of a specified text object.

Syntax: **Text_SetString**(*TX_ref, Text*)
TX_Ref: a numeric expression representing the object identification number
Text: text variable or text expression within quotes
Returns: 0 if successful, or error code if operation not successful

Usage:
```
Value2 = Text_SetString(1, "New String") ;  {changes text for obj 1}
```

Text_SetStyle

Changes the horizontal and vertical position style for the specified text object.

Syntax: **Text_SetStyle**(*TX_ref, Horiz,Vert*)
TX_Ref: a numeric expression representing the object identification number
Horiz: 0 for left, 1 for right, 2 for center
Vert: 0 for top, 1 for bottom, 2 for center
Returns: 0 if successful, or error code if operation not successful

Usage:
```
Value1 = Text_SetStyle(3,0,1) ;       {repositions obj 3 to the left-bottom}
```

Than

Skip word used to improve readability. Ignored during execution.

Usage:
```
If High > than the Highest(Close, 14) Then {any operation}
```

The

Skip word used to improve readability. Ignored during execution. *(see Than)*

Then

Precedes the operation(s) to be executed when the matching *If* condition is true.

```
Usage:  If Condition1 Then Begin
            {Operations done if condition is true}
        End ;
```

This

Used to reference the current Bar.

```
Usage:  Buy This Bar on Close;
```

Thursday

Specifies day of the week Thursday (numeric value = 4).

Ticks

Reserved for backward compatibility. Replaced with Points.

TickType

The kind of tick that triggered an option core event: Asset, Option, Future, or Model.

Time

Closing time of the current bar in 24-hour HHMM format.

```
Usage:  Value1 = Time;                    {returns 2130 if the bar time is 9:30pm}
```

TL_Delete

Deletes the specified trendline from the chart.

Syntax: **TL_Delete**(*TL_Ref*)

TL_Ref: a numeric expression representing the trendline identification number

Returns: 0 if operation is successful, or error code if not

```
Usage1: TL_Delete(2);                              {deletes trendline number 2}
Usage2: Value1 = TL_Delete(3);      {returns status code after deleting trendline 3}
```

TL_GetAlert

Gets the alert status of the specified trendline object.

TL_Ref: a numeric expression representing the trendline identification number

Returns: 0 = no alert, 1 = Breakout Intrabar, 2 = Breakout on Close

```
Usage:  Value2 = TL_GetAlert(4);          {returns alert status for  trendline number 4}
```

TL_GetBeginDate

The date of the starting point for the specified trendline. (see **TL_Delete** for syntax)

```
Usage:  Value1 = TL_GetBeginDate(2);      {returns the start date of  trendline 2}
```

TL_GetBeginTime

The time of the starting point for the specified trendline. (see **TL_Delete** for syntax)

```
Usage:  Value2 = TL_GetBeginTime(3);      {returns the start time of  trendline 3}
```

TL_GetBeginVal

The price (vertical axis) of a trendline's starting point. (see **TL_Delete** for syntax)

Usage: `Value1 = TL_GetBeginVal(4);` {returns the start price of trendline 4}

TL_GetColor

Returns the color value of the specified trendline. (see **TL_Delete** for syntax)

Usage: `Value1 = TL_GetColor(3);` {returns the color of trendline 3}

TL_GetEndDate

The date of the ending point for the specified trendline. (see **TL_Delete** for syntax)

Usage: `Value1 = TL_GetEndDate(2);` {returns the end date of trendline 2}

TL_GetEndTime

The date of the ending point for the specified trendline. (see **TL_Delete** for syntax)

Usage: `Value2 = TL_GetEndTime(4);` {returns the end time of trendline 4}

TL_GetEndVal

The price (vertical axis) of a trendline's ending point. (see **TL_Delete** for syntax)

Usage: `Value1 = TL_GetEndVal(3);` {returns the end price of trendline 3}

TL_GetExtLeft

True if the specified trendline is extended left, False otherwise. (see **TL_Delete** for syntax)

Usage: `Condition1 = TL_GetExtLeft(12) ;` {true if trendline 12 extends left}

TL_GetExtRight

True if the specified trendline is extended right, False otherwise. (see **TL_Delete** for syntax)

Usage: `Condition1 = TL_GetExtRight(5) ;` {true if trendline 5 extends right}

TL_GetFirst

Returns the ID number for the first trendline of a specified type.

Syntax: **TL_GetFirst**(*Type*)

Type: identifies the origin of the requested first trendline

1 = trendline created by an analysis technique

2 = trendline created by the drawing object only, and

3 = trendline created by either the drawing object or an analysis technique

Returns: ID if operation successful or error code if not

Usage: `Value2 = TL_GetFirst(2);` {returns id of first trendline of type}

TL_GetNext

Returns the text object ID for the next object of a specified type after specified object.

Syntax: **TL_GetNext**(*TL_Ref*, *Type*)

TL_Ref: a numeric expression representing the object identification number

Type: (see TL_GetFirst)

Returns: ID if operation successful or error code if not

Usage: `Value1 = TL_GetNext(2,1);` {returns id of trendline draw object after id 2}

TL_GetSize

The line thickness setting (weight) for the specified trendline. (see **TL_Delete** for syntax)

Usage: `Value2 = TL_GetSize(3);` {returns thickness of trendline 3}

TL_GetStyle

The line style for the specified trendline.

Syntax: **TL_GetStyle**(*TL_Ref*)

TL_Ref: a numeric expression representing the object identification number

Returns: Tool_Solid = 1 (solid)
Tool_Dashed = 2 (dashed)
Tool_Dotted= 3 (dotted)
Tool_Dashed2= 4 (dashed pattern)
Tool_Dashed3= 5 (dashed pattern)

Usage: `Value1 = TL_GetStyle(6);` {returns 3 if trendline 6 is dotted}

TL_GetValue

The price (vertical axis) of the specified trendline at date and time.

Syntax: **TL_GetStyle**(*TL_Ref,cDate,Time*)

TL_Ref: a numeric expression representing the object identification number
cDate: date in YYYMMDD format
Time: time in HHMM 24-hour format
Returns: price if operation successful or error code if not

Usage: `Value2 = TL_GetValue(2,991104,0930);` {returns the price of trendline 2}

TL_New

Creates a new trendline using specified start and end points.

Syntax: **TL_New**(*sDate,sTime,sPrice,eDate,eTime,ePrice*)

sDate: starting point date in YYYMMDD format
sTime: starting point time in HHMM 24-hour format
sPrice: starting point price
eDate: ending point date in YYYMMDD format
eTime: ending point time in HHMM 24-hour format
ePrice: ending point price
Returns: trendline ID if operation successful, error code if not

Usage: `Value1 = TL_New(990107, 0930, 45, 990125, 1600, 37.250);`

TL_SetAlert

Sets the alert status for a specified trendline.

Syntax: **TL_SetAlert**(*TL_Ref, Status*)

TL_Ref: a numeric expression representing the object identification number
Status: 0=no alert, 1=breakout intrabar alert, 2=breakout on close alert

Usage: `TL_SetAlert(3, 1);` {sets intrabar alert for trendline 3}

TL_SetBegin

Changes the starting point of a specified trendline.

Syntax: **TL_SetBegin**(*TL_Ref*,*sDate*,*sTime*,*sPrice*)
TL_Ref: a numeric expression representing the object identification number
(see **TL_New** for descriptions of *sDate*,*sTime*,*sPrice*)

Usage:
```
TL_SetBegin(4, 990221, 1015, 107.225);
```

TL_SetColor

Changes the color of a specified trendline.

Syntax: **TL_SetColor**(*TL_Ref*, *Color*)
TL_Ref: a numeric expression representing the object identification number
Color: the color name or numeric value

Usage:
```
TL_SetColor(3, Blue);          {sets trendline 3 to color blue}
```

TL_SetEnd

Changes the ending point of a specified trendline.

Syntax: **TL_SetEnd**(*TL_Ref*, *eDate, eTime, ePrice*)
TL_Ref: a numeric expression representing the object identification number
(see **TL_New** for descriptions of *eDate*,*eTime*,*ePrice*)

Usage:
```
TL_SetEnd(2, 990221, 1515, 207.125);
```

TL_SetExtLeft

Changes the leftward extension status of a specified trendline.

Syntax: **TL_SetExtLeft**(*TL_Ref*, *Status*)
TL_Ref: a numeric expression representing the object identification number
Status: True turns on leftward extension, False turns it off

Usage:
```
TL_SetExtLeft(2,True);         {turns on left extend for trendline 2}
```

TL_SetExtRight

Changes the rightward extension status of a specified trendline.

Syntax: **TL_SetExtRight**(*TL_Ref*, *Status*)
TL_Ref: a numeric expression representing the object identification number
Status: True turns on rightward extension, False turns it off

Usage:
```
TL_SetExtRight(3,False);       {turns off right extend for trendline 3}
```

TL_SetSize

Changes the line thickness setting (weight) for the specified trendline.

Syntax: **TL_SetSize**(*TL_Ref*, *Size*)
TL_Ref: a numeric expression representing the object identification number
Size: numeric value ranging from 0 (the thinnest) to 6 (the thickest).

Usage:
```
TL_SetSize(2,4);               {sets trendline 2 to thickness 4}
```

TL_SetStyle

Changes line style for the specified trendline.

Syntax: **TL_SetStyle**(*TL_Ref,Type*)

TL_Ref: a numeric expression representing the object identification number

Type: Tool_Solid = 1 (solid)
Tool_Dashed = 2 (dashed)
Tool_Dotted= 3 (dotted)
Tool_Dashed2= 4 (dashed pattern)
Tool_Dashed3= 5 (dashed pattern)

Usage:
```
TL_SetStyle(4,Tool_Dashed);          {sets trendline 4 to dashed}
```

To

Used in a For-Loop statement to separate the starting and ending counter values.

Usage:
```
For Value5 = Start To Start + 10 Begin
      {Any operations}
End ;
```

Today

Retained for backward compatibility. Replaced by **This Bar**.

Tomorrow

Retained for backward compatibility. Replaced by **Next Bar**.

Tool_Black

Retained for backward compatibility. Replaced by the color name **Black**.

Tool_Blue

Retained for backward compatibility. Replaced by the color name **Blue**.

Tool_Cyan

Retained for backward compatibility. Replaced by the color name **Cyan**.

Tool_DarkBlue

Retained for backward compatibility. Replaced by the color name **DarkBlue**.

Tool_DarkBrown

Retained for backward compatibility. Replaced by the color name **DarkBrown**.

Tool_DarkCyan

Retained for backward compatibility. Replaced by the color name **DarkCyan**.

Tool_DarkGray

Retained for backward compatibility. Replaced by the color name **DarkGray**.

Tool_DarkGreen

Retained for backward compatibility. Replaced by the color name **DarkGreen**.

Tool_DarkMagenta

Retained for backward compatibility. Replaced by the color name **DarkMagenta**.

Tool_DarkRed

Retained for backward compatibility. Replaced by the color name **DarkRed**.

Tool_DarkYellow

Retained for backward compatibility. Replaced by the color name **DarkYellow**.

Tool_Dashed

Represents a dashed line style (2) used with drawing objects.

Tool_Dashed2

Represents a dashed line style (4) used with drawing objects.

Tool_Dashed3

Represents a dashed line style (5) used with drawing objects.

Tool_Dotted

Represents a dotted line style (3) used with drawing objects.

Tool_Green

Retained for backward compatibility. Replaced by the color name **Green**.

Tool_LightGray

Retained for backward compatibility. Replaced by the color name **LightGray**.

Tool_Magenta

Retained for backward compatibility. Replaced by the color name **Magenta**.

Tool_Red

Retained for backward compatibility. Replaced by the color name **Red**.

Tool_Solid

Represents a solid line style (1) used with drawing objects.

Tool_White

Retained for backward compatibility. Replaced by the color name **White**.

Tool_Yellow

Retained for backward compatibility. Replaced by the color name **Yellow**.

Total

Specifies the number of shares/contracts to exit from a position created by pyramiding.

Usage:
```
Sell 5 Contracts Total Next Bar at Market;
```
{Exits five contracts/shares from the entire long position}

TotalBarsLosTrades

The total number of bars that elapsed during losing trades for all closed trades.

Usage: `Value2 = TotalBarsLosTrades ;`

TotalBarsWinTrades

The total number of bars that elapsed during winning trades for all closed trades.

Usage: `Value1 = TotalBarsWinTrades ;`

TotalTrades

The total number of closed trades in the current strategy.

TrailingStopAmt

Retained for backward compatibility with previous versions of the product. Replaced by the reserved word **SetDollarTrailing**.

TrailingStopFloor

Retained for backward compatibility with previous versions of the product. Replaced by the reserved word **SetPercentTrailing**.

TrailingStopPct

Retained for backward compatibility with previous versions of the product. Replaced by the reserved word **SetPercentTrailing**.

True

Represents a true, or correct, conditional expression.

TrueFalse

Defines an input that expects a true/false expression.

Usage: `Input: Switch(TrueFalse);` {accepts a true/false input for Switch}

TrueFalseArray

Defines an input that expects a true/false expression for each array element.

Usage: `Input: MyArray[n](TrueFalseArray)` {accepts t/f inputs by value}

TrueFalseArrayRef

Defines an input that expects a true/false variable reference for each array element.

Usage: `Input: MyArray[n](TrueFalseArrayRef)` {accepts t/f inputs by reference}

TrueFalseRef

Defines an input that expects a true/false variable reference.

Usage: `Input: Switch(TrueFalseRef)` {accepts a t/f input by reference}

TrueFalseSeries

Defines an input as a true/false series expression.

Usage: `Input: Flag(TrueFalseSeries);` {accepts a t/f input with history}

TrueFalseSimple

Defines an input as a true/false simple expression.

```
Usage:  Input: Switch(TrueFalseSimple);        {accepts a t/f input without history}
```

TtlDbt_By_NetAssts

Returns the total debt (long + short term) divided by total assets.

Tuesday

Specifies day of the week Tuesday (numeric value = 2).

Under

Used only with **Crosses** to detect a value crossing under, or below, another value.

```
Usage:  If Value1 Crosses Under Value2 Then {Any Operation} ;
```

UnionSess1EndTime

Latest session 1 end time of all data in a multi-data chart.

UnionSess1FirstBar

Earliest session 1 first bar time of all data in a multi-data chart.

UnionSess1StartTime

Earliest session 1 start time of all data in a multi-data chart.

UnionSess2EndTime

Latest session 2 end time of all data in a multi-data chart.

UnionSess2FirstBar

Earliest session 2 first bar time of all data in a multi-data chart.

UnionSess2StartTime

Earliest session 2 start time of all data in a multi-data chart.

Units

Retained for backward compatibility.

UNSIGNED

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

Until

Reserved for future use.

UpperStr

Used to convert a string expression to uppercase letters.

```
Usage1:  Value1 = UpperStr("My TextString");     {returns "MY TEXTSTRING"}
```

UpTicks

Number of ticks on a bar whose value is higher than the tick immediately preceding it.

V

Abbreviation for Volume. Returns the volume of shares/contracts of a referenced bar.

Usage: **If** *MyVol* > **V** of 1 **Bar Ago Then SellShort** at **Close**;

Var

Declares a variable name to use throughout your analysis technique. Shorthand form.

Usage:
```
Var: Count(10);          {declares the variable Count with an initial value of 10}
```

Variable

Declares a variable name to use throughout your analysis technique.

Usage:
```
Variable: Val(5);        {declares the variable Val with an initial value of 5}
```

Variables

Declares multiple variable names separated by commas.

Usage:
```
Variables: Countup(0),Countdown(10); {declares and initializes variables}
```

Vars

Declares multiple variable names separated by commas. Shorthand form.

Usage:
```
Vars: MyVal(2), MyPrice(31);          {declares and initializes variables}
```

VARSIZE

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

VARSTARTADDR

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

VOID

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

Volume

Returns the number of shares/contracts traded for the referenced bar.

Usage: **If** *TestVol* > **Volume** of 3 **Bars Ago Then Buy** at **Market**;

Was

Skip word ignored during execution.

Usage: **If Close** was < than the **Lowest**(**Close**, 14) **Then** *{any operation}* ;

Wednesday

Specifies day of the week Wednesday (numeric value = 3).

While

Defines instructions that are executed until a true/false expression returns *False*.

Usage:
```
While Condition1 Begin
      {any operations};
End; {continues to loop until the Condition is no longer true}
```

White

Specifies color White (numeric value = 8) for plots and backgrounds.

WORD

Reserved for use with custom DLLs designed for EasyLanguage and ELKIT32.DLL.

Year

Year on specified calendar date, in short form (last 2 or 3 digits of year)

Returns the year (YYY) portion of the specified calendar date.

Syntax: **Year**(c*Date*)*;*

cDate: numeric expression for the date in YYMMDD or YYYMMDD format.

Usage:
```
Value1 = Year(1011004)          {returns the year 101 representing 2001}
```

Yellow

Specifies color Yellow (numeric value = 7) for plots and backgrounds.

Yesterday

Retained for backward compatibility. Refers to the previous bar.

About the CD-ROM

INTRODUCTION

The files on the enclosed CD-ROM are TradeStation2000i and TradeStation 6.0 compatible EasyLanguage source code.

System Requirements

Processor:	300 MHz Pentium or better
Hardware:	1024×768 monitor resolution or higher CD-ROM drive
Software:	Windows 95/98/Me/NT/2000/XP
RAM:	128 MB RAM
Hard Drive:	60 MB free space
Software:	TradeStation 4.0 or TradeStation 2000i or TradeStation 6.0 Microsoft Internet Explorer 5.0 or higher
Internet Connection:	Dial-Up (56K bps) or Direct (ISDN, Cable Modem, DSL, T1)

USING THE FILES

Loading Files

To use the files, launch TradeStation and use the **Import** feature from within the TradeStation PowerEditor to import the EasyLanguage source code. Use the files according to your needs.

Printing Files

If you want to print the files, select File, Print from the pull-down menu.

Saving Files

When you have finished editing a file, you should save it under a new file name by selecting **File, Save As** from the pull-down menu.

USER ASSISTANCE

If you have a damaged CD-ROM, please contact Wiley Technical Support at:

Phone: 201-748-6753
Fax: 201-748-6450 (Attention: Wiley Technical Support)
URL: www.wiley.com/techsupport
Email: techhelp@wiley.com

If you need assistance with using the files on the CD-ROM or questions about the source code, please contact George Pruitt at info@futurestruth.com.

Any questions regarding TradeStation™ products should be forwarded to TradeStation Securities at www.tradestation.com.

To place additional orders or to request information about other Wiley products, please call (800) 225-5945.

Index

K

L

M

N

R

S

Y