

By Justin Kuepper

<http://www.investopedia.com/university/systemcoding/>

Thanks very much for downloading the printable version of this tutorial.

As always, we welcome any feedback or suggestions.

<http://www.investopedia.com/contact.aspx>

Table of Contents

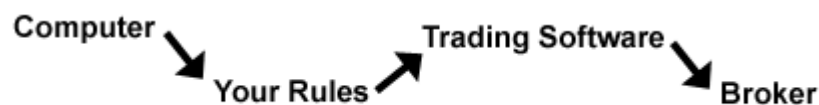
- 1) Trading Systems Coding: Introduction
- 2) Trading Systems Coding: System Design
- 3) Trading Systems Coding: The Coding Stage
- 4) Trading Systems Coding: The Coding Process
- 5) Trading Systems Coding: Testing, Troubleshooting and Optimizing
- 6) Trading Systems Coding: Using Your System
- 7) Trading Systems Coding: Conclusion

Introduction

Trading systems are simply sets of rules that traders use to determine their entries and exits from a position. Developing and using trading systems can help traders attain consistent returns while limiting risk. In an ideal situation, traders should feel like robots, executing trades systematically and without emotion. So, perhaps you've asked yourself: What's to stop a robot from trading my system? The answer: Nothing! This tutorial will introduce you to the tools and techniques that you can use to create your own automated trading system.

How Are Automated Trading Systems Created?

Automated trading systems are created by converting your trading system's rules into code that your computer can understand. Your computer then runs those rules through your trading software, which looks for trades that adhere to your rules. Finally, the trades are automatically placed with your broker.



Copyright © 2006 Investopedia.com

This tutorial will focus on the second and third parts of this process, where your rules are converted into a code that your trading software can understand and use.

What Trading Software Supports Automated Trading Systems?

There are many trading programs that support automated trading systems. Some will automatically generate and place trades with your broker. Others will automatically find trades that fit your criteria, but require that you place the orders with your broker manually. Moreover, fully automatic trading programs often require that you use specific brokerages that support such features; you may also have to complete an additional authorization form.

Fully Automatic	Semi Automatic
TradeStation	AmiBroker
Interactive Brokers	Tradecision
WealthLab	

Advantages and Disadvantages

Automated trading systems have several benefits, but they also have their downsides. After all, if someone had a trading system that automatically made money all the time, he or she would literally own a money making machine!

Advantages:

- An automated system takes the emotion and busy-work out of trading, which allows you to focus on improving your strategy and [money management](#) rules.
- Once a profitable system is developed, it requires no work on your part until it breaks, or market conditions demand a change.

Disadvantages:

- If the system is not properly coded and tested, large losses can occur very quickly.
- Sometimes it is impossible to put certain rules into code, which makes it difficult to develop an automated trading system.

In this tutorial you will learn how to plan and design an automated trading system, how to translate this design into code that your computer will

understand, how to test your plan to ensure optimal performance and, finally, how to put your system to use.

System Design

The first step when coding any application is the design phase. Whether coding a software application or a trading system, careful design and planning will help you finish in a shorter amount of time with fewer errors. We will be using a simple three-step process to design our trading system.

Step 1: Create Your Trading System Rules

The first step when designing a trading system is simply coming up with the rules by which your system will operate. There should be four core rules to every trading system:

1. Buy - Identify when you want to buy a position.
2. Sell - Identify when you want to sell a position.
3. Stop - Identify when you want to cut your losses.
4. Target - Identify when you want to book a gain.

So, for example:

1. Buy - When the 30-day [moving average](#) (MA) crosses above the 60-day MA
2. Sell - When the 30-day MA crosses below the 60-day MA
3. Stop - Maximum loss of 10 units
4. Target - Target of 10 units

This example system will buy and sell based on the 30- and 60-day moving averages and will automatically book gains after a 10-unit profit or sell at a loss after a 10-unit move in the opposite direction.

Step 2: Identify the Components of Each Rule

Now that we have our rules down, we need to identify the components involved in each rule. Each component should contain two elements:

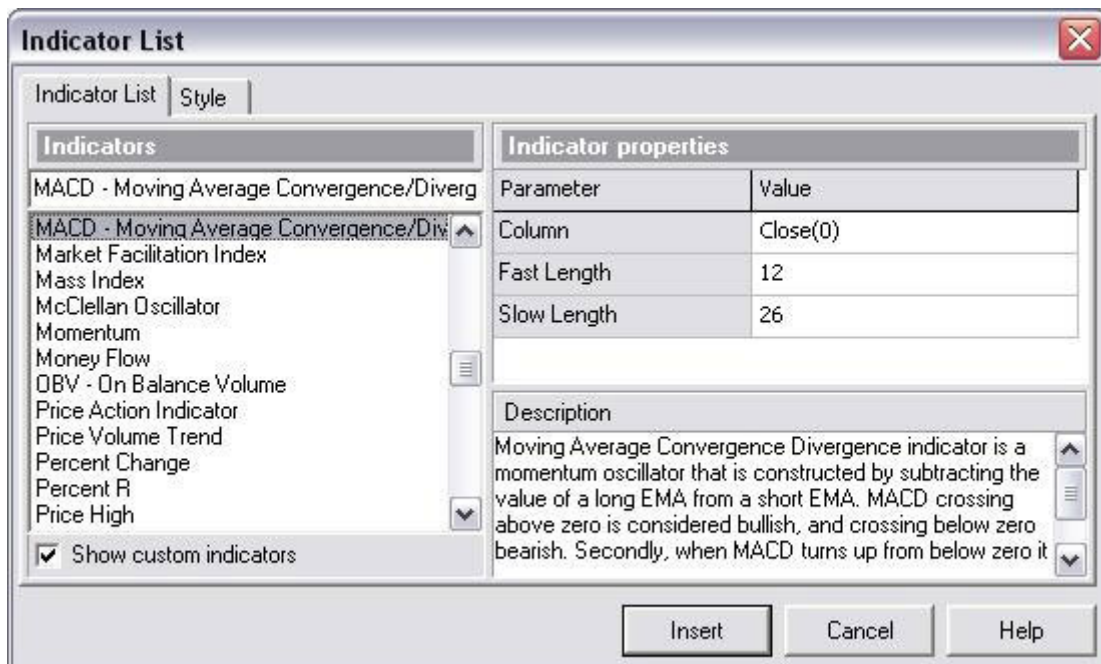
1. The indicator or study used
2. The settings for the indicator or study

These components should be constructed by typing the shorthand name for the study, followed by the settings in parentheses. These settings in parentheses are referred to as "parameters" of the indicator or study. Occasionally, a study may have multiple parameters, in which case you simply separate them with commas.

Let's take a look at a few examples:

1. MA(25) - 25-day moving average
2. RSI(25) - 25-day [relative strength index](#)
3. MACD(Close(0),5,5) - [Moving average convergence divergence](#) set based on today's close, with a five-day fast length and a five-day slow length

If you are unsure of how many parameters a certain component requires, you can simply consult your trading program's documentation, which lists these components along with the values that need to be filled in. For example, we can see that Tradecision tells us that we need three parameters with MACD:



Source:MetaTrader

Copyright © 2006 Investopedia.com

So, for the example mentioned in step one, we would use:

1. MA(30) - Meaning 30-day moving average
2. MA(60) - Meaning 60-day moving average

Step 3: Adding Action

Now we will add actions to our rules. Each action adheres to the following basic format:

IF Condition [**WHILE** Condition] **THEN** Action

Typically, the condition will consist of the components and parameters you created above, while the action will consist of buy or sell. Conditions may also consist of simple English if no component is present. Note that the "while" component is optional.

Here are a few examples to help illustrate this point:

- **IF** MA(30) Crosses Above MA(60) **THEN** Buy
- **IF** MA(30) Crosses Below MA(60) **WHILE** Volume(20,000) **THEN** Sell
- **IF** EMA(25) Is Greater Than MA(5) **THEN** Sell
- **IF** RSI(20) Is Equal To 50 **THEN** Buy

So, for the example we've been using, we'd simply list:

- IF MA(30) Crosses Above MA(60) THEN Buy
- IF MA(30) Crosses Below MA(60) THEN Sell
- IF our trade has 10 units of profit THEN Sell
- IF our trade has 10 units of loss THEN Sell

What's Next?

Next, we'll take a look at converting these rules into a code that your computer can understand! your profits.

The Coding Stage

Now that we have a design document in hand, we can look at how these rules are put into code that a computer can understand. In this section, we'll break down a section of code and look at it piece by piece. For this example, we will use MetaTrader's programming language MetaQuotes II to build a very simple [moving average](#) trading system.

```
Defines: MATrendPeriod(100);
```

```
Var:MaCurrent(0),MaPrior(0);
```

```
If Bars < 100 Then Exit;
```

```
If FreeMargin < 1000 then Exit;
```

```
maCurrent =iMA(MATrendPeriod,MODE_SMA,0);
```

```
maPrior    =iMA(MATrendPeriod,MODE_SMA,1);
```

```
If maCurrent > maPrior then
```

```
{SetOrder(OP_BUY,Lots,Ask,3,0,Ask+TakeProfit*Point,RED); Exit;};
```

```
If maCurrent < maPrior then
```

```
{SetOrder(OP_SELL,Lots,Bid,3,0,Bid-TakeProfit*Point,RED);Exit;};
```

The If/Then Format

After taking a brief look at this code, you should recognize a few elements that we touched on in the design phase. For example, you should recognize the If/Then format that we used when constructing our design. To get a better understanding of this code, let's break it down and analyze each part:

```
Defines: MATrendPeriod(100);
```

```
Var: MaCurrent(0),MaPrior(0);
```

Here, we're simply defining the moving average that we are using by saying that we want it to be an average of the last 100 [bars](#). The **Defines** function here lets us do that for any type of data we want. After that, we simply invent two variables (items which we create to hold data) **MaCurrent(0)** and **MaPrior(0)**. These two variables will hold the data that we will set in a later step.

```
If Bars < 100 Then Exit;
```

```
If FreeMargin < 1000 then Exit;
```

Here we see the If/Then format that we used in the design phase put to work. These two statements tell the computer to exit if certain conditions aren't met. Let's translate these two commands into English:

If Bars < 100 Then Exit; → "If there are fewer than 100 bars (data points) on the chart, then exit the program without doing anything."

If FreeMargin < 1000 then Exit; → "If my account has less than \$1,000 in available funds, then exit the program without doing anything."

You can translate any criteria you want into this format and put it at the beginning

of your program in order to adapt to certain situations.

Buy and Sell Signals

```
maCurrent =iMA(MATrendPeriod,MODE_SMA,0);
```

```
maPrior =iMA(MATrendPeriod,MODE_SMA,1);
```

Now let's make use of the two variables we described above. Let's take these statements apart to see what they are doing:

maCurrent = → Here we are telling the computer to assign the following information to "maCurrent".

iMA(MATrendPeriod,MODE_SMA,0); → Here we are using a simple statement, which uses the following basic format: Study(TimePeriod,Mode,Start).

Note that you can replace iMA with [MACD](#), [RSI](#) or any other studies your trading system may be using. You can also replace the parameters to suit your own system.

```
If maCurrent > maPrior then
```

```
{SetOrder(OP_BUY,Lots,Ask,3,0,Ask+TakeProfit*Point,RED); Exit;};
```

Now we are getting somewhere! Here is the part of the code that tells the computer when to buy. Notice that we are making use again of the If/Then format that we used in the design document. Let's translate this to English to see what's happening:

If maCurrent > maPrior then { → "If the current Moving Average is greater than the prior Moving Average, then..."

SetOrder(→ "Create an order entry to ..."

OP_BUY,Lots,Ask,3,0,Ask+TakeProfit*Point,RED → "Buy my defined number of lots at the ask price plus my take profit point, and mark it as a red point on the chart."

); → "End the order."

Exit; → "Exit the trading strategy."

}; → "End the If/Then statement."

Note that the Take Profit Point is something that is defined by users when they add the trading system to their charts. Also notice that we are buying at the [ask](#) price and selling at the [bid](#) price - this is a key feature, especially when creating a system for stocks.

If `maCurrent < maPrior` then

```
{SetOrder(OP_SELL,Lots,Bid,3,0,Bid-TakeProfit*Point,RED); Exit;};
```

Finally, we have our code that tells the computer when to take a [short position](#). Note that this statement is almost identical to the "buy" statement, aside from the `OP_SELL` instead of `OP_BUY` as well as the usage of the bid price as opposed to the ask price.

Conclusion

And there you have it - the bare bones of what a trading system code looks like. Please note that the above code is not a complete trading system, as it does not include any commands to close open positions. Such additional aspects can be implemented using a format similar to the code we've shown.

In the next section of this tutorial, we will go into greater depth regarding the specific ways in which your trading system can be converted to code.

The Coding Process

By now you should have a design in hand as well as a basic idea of what the code looks like. In this section, we'll take a more in-depth look at how a program is created. After reading this section, you should be able to understand basic program structure and be able to convert your design to code!

An Overview

There are two basic parts to a program:

- **Variables** - These are items that hold data. This can be data that you collect from the user, or any other data.
- **Statements** - These form the core of the program. Statements manipulate the data to get results that can be converted to actions.

In addition to these core components, there are also several optional components:

- **Functions** - These are simply collections of related statements that can be used to perform a specific task. For example, a function telling you when to buy might include a statement to check whether you have enough money, a statement to determine whether it meets your criteria and a statement to place the order. A function combines these, allowing you to simply call on the function instead of rewriting these statements each time you want to buy. |
- **Arrays** - These are simply data structures that hold similar data and enable you to access and manipulate the data more efficiently.

A Look at Variables

Variables are simply objects that you define to hold data. You may recall from the previous section that we used three variables: MATrendPeriod, MaCurrent and MaPrior. MATrendPeriod held a number that defined how many days we would use in our moving average calculation; MaCurrent held a number representing the current moving average; and MaPrior held a number representing the prior moving average.

Creating a Variable

You can use almost any name you want when naming a variable. The only exception is a list of "restricted words" that you are not allowed to use because the names are already used by other parts of the program. You can find your trading program's list of restricted words in the program's documentation. In general, names should describe the data being held. For example, notice that we used MaCurrent to define the current moving average.

After you have created a name, you must declare and define the variable. Declaring a variable tells the computer what type of data it is, and tells it to make space for that data. Defining the variable is where actual data is assigned, or added, to the variable. Let's take a look at these processes:

1. *Declaring a Variable*

In MetaTrader, variables are declared automatically when you assign information to them. In other programs, you may have to declare a variable, which is typically done using the following format:

`<data type> <variable name>;`

The two types of data are numbers and text, but these are broken down into more groups like integers (whole numbers), double (large numbers), float

(decimal numbers), string (text), and others depending on the program you are using. For example, the following code will declare numberOfDays as an integer:

```
Int numberOfDays;
```

2. Defining a Variable

After your variable has been declared, the computer has created space for it. Now, all you have to do is add actual data to that space. This can be done in two ways: you can either define a set amount, or you can perform a calculation to obtain a value, which you then assign to the variable.

In MetaTrader, you can add set data using the following format:

```
Defines: <variable name>(<set amount>);
```

In other programs, set data is often assigned simply using the equals sign:

```
<variable name> = <set amount>;
```

If you want to perform a calculation to obtain data to assign to the variable, then you simply assign the variable to the calculation:

```
<variable name> = <calculation>;
```

For example, to set a 20-day moving average in MetaTrader, we use the following code:

```
<variable name> = iMA(20,MODE_SMA,0);
```

Note that the `iMA(20,MODE_SMA,0)` portion of the code is the calculation. The format for this calculation was developed by MetaTrader and will differ if you are using another trading program. To find these calculations, you must consult your trading program's documentation, which usually contains a list of all available calculations.

3. Using Variables

Once declared and defined, variables can be used anywhere else within the program to represent the data they contain. To do this, simply type the name of the variable in place of the data. For example, if MATrendPeriod contains the number of days we want a moving average calculated for, we can use it to replace the 20 in our example above:

```
<variable name> = iMA(MATrendPeriod, MODE_SMA, 0);
```

There are two advantages to using variables as opposed to just the data: (1) you can change the data in one place, and (2) the result of an entire calculation can be contained within one variable.

A Look at Statements

Statements are the core of any program - they contain all of the commands that manipulate data to make decisions. Here we will take a look at several of the most common types of statements and how they can be used.

1. Comments

If you have designed a complex trading system, it may take a lot of code to implement your rules; therefore, it would be prudent to insert comments in your code to help yourself understand it in the future, and to help out anyone with whom you may share your code. Almost all trading applications share a similar method for creating comments:

Single Line Comments:

```
//<your comment here>
```

Multi-Line Comments:

```
/* <comment line one>  
<comment line two>  
<comment line three> */
```

2. The 'If' Statement

This is the statement you will use most when coding a trading system. This statement lets you create scenarios as we did in the design portion of this tutorial. You may have also noticed that this was the only statement we used in the example program we created. This type of statement is implemented using the following format:

Standard If/Then:

```
If <condition> Then <action>;
```

Standard If/Else:

```
If <condition> Then <action> Else <action>;
```

So, for example:

```
If accountBalance < 200 then Exit;
```

Note that the conditional part of the 'If' statement is constructed using the following:

`<object one> <condition> <object two>`

The condition can be:

- Greater Than (>)
- Less Than (<)
- Equal To (=). Note that one '=' assigns, two '==' returns either true or false.

3. *The While Loop*

This loop is commonly used to tell the computer to continue doing something while a certain condition is true or false. So, for example, maybe you want to have the trading system keep a position open while your account is above a certain balance, but close it if it ever falls below that balance. These statements are created using the following format:

`While <condition> <action>;`

4. *The Exit and/or End Statement*

An 'Exit' or 'End' statement is used to indicate to the computer that your program will be ending at that particular point. Typically, this is done using:

`Exit;`

Or,

`End;`

These are usually placed in the 'If' statements if that statement is executed so that the computer doesn't continue to look at the rest of the 'If' statements.

Trading System Implementation

Note that different trading applications will differ slightly in how they implement statements. For example, in some trading applications, the 'If' statement is constructed by using:

`If (<condition>, <then do this>, <else do this>);`

Meanwhile, other applications may split up this code into two parts:

`If (<condition>) { <then do this> } else { <else do this> };`

We can see that the same idea is present, but the implementation differs. It is

important to consult your trading application's documentation, or [application programming interface](#) (API), to determine what differences exist.

Putting It All Together

Now you should have an idea of the different components that can be used when coding your trading system. All that remains to be done is to put everything together. To do this, simply take your design document and determine the following:

- What variables will I have to define?
 - Calculations → Moving averages, RSI, MACD, etc.
 - Set Amounts → Time periods, deposit amounts, risk percentages, etc.
- What statements will I have to make?
 - Convert your rules to the proper statements using the above guides.

Once you know this, all you need to do is piece together all of the parts. The standard structure for a program is:

```
<includes>  
<variables>  
<statements>
```

Conclusion

Now you should have a basic idea of how to put your trading system into code. Be sure to consult your trading application's documentation often, as it may contain pre-built calculations that you can use, code examples and much more that can help you to better understand the specifics. In the next part of this tutorial, we'll take a look at testing your new program both technically (to find errors in the code) and theoretically (to find errors in your logic).

Testing, Troubleshooting and Optimizing

Now that you have a trading system designed and coded, it is time to test it to make sure that your coding is free of logical and technical errors. We will also look at something known as optimization - a feature in some trading programs that allows you to fine tune your trading rules to fit the stocks that you plan on trading.

Testing Your Trading System

The vast majority of trading applications that support programming languages

also support testing tools. These tools are divided into two categories:

1. Technical

Technical testing tools search for technical errors in your code. For example, if you forget to add a semicolon after a statement, the technical testing tool will notify you that your statement is invalid.

The location of the technical testing tool depends on the trading application being used. MetaTrader displays an error or flawed results when you try to compile your code, while trading applications like Tradecision have a "code check" utility built into the interface that lets you check your code for errors before applying it.

2. Logical

Logical testing tools search for logical errors in your code. For example, if you happened to use a "greater than" sign instead of a "less than" sign (which is not a technical error), a logical testing tool will show you that your results don't make sense.

The most popular logical testing tool is the [backtesting](#) tool. This tool allows you to take past data and apply your trading system to that data. This gives you an idea of the following:

- Whether your trading system is a profitable one
- What conditions prove to be most profitable
- Where any errors in your rules might exist

(For more information, see [Backtesting: Interpreting The Past.](#))

Troubleshooting Your Trading System

As with any other type of programming, troubleshooting can be a tedious and difficult task. Finding errors in your code requires systematically sorting through your code to identify syntactical errors that, although often minor, can bring your program to a halt.

Here are some common errors to look for:

- Missing semicolons after statements - These have to be after every statement.
- Undefined variables - Remember that you have to declare them before you use them!
- Spelling mistakes - If any names or functions are spelled incorrectly, the trading application will return an error (see example below).
- Incorrect usage of (=) - Remember that "=" assigns one value to another value, while "==" means "equal to".
- Incorrect usage of built-in functions - Consult your trading application's documentation or [application programming interface](#) (API) to make sure that you are using the correct syntax.

Some trading applications contain a feature that will let you test your code before using or compiling it. This feature allows you to see what the error is and on which line it can be found. Take Tradecision for example

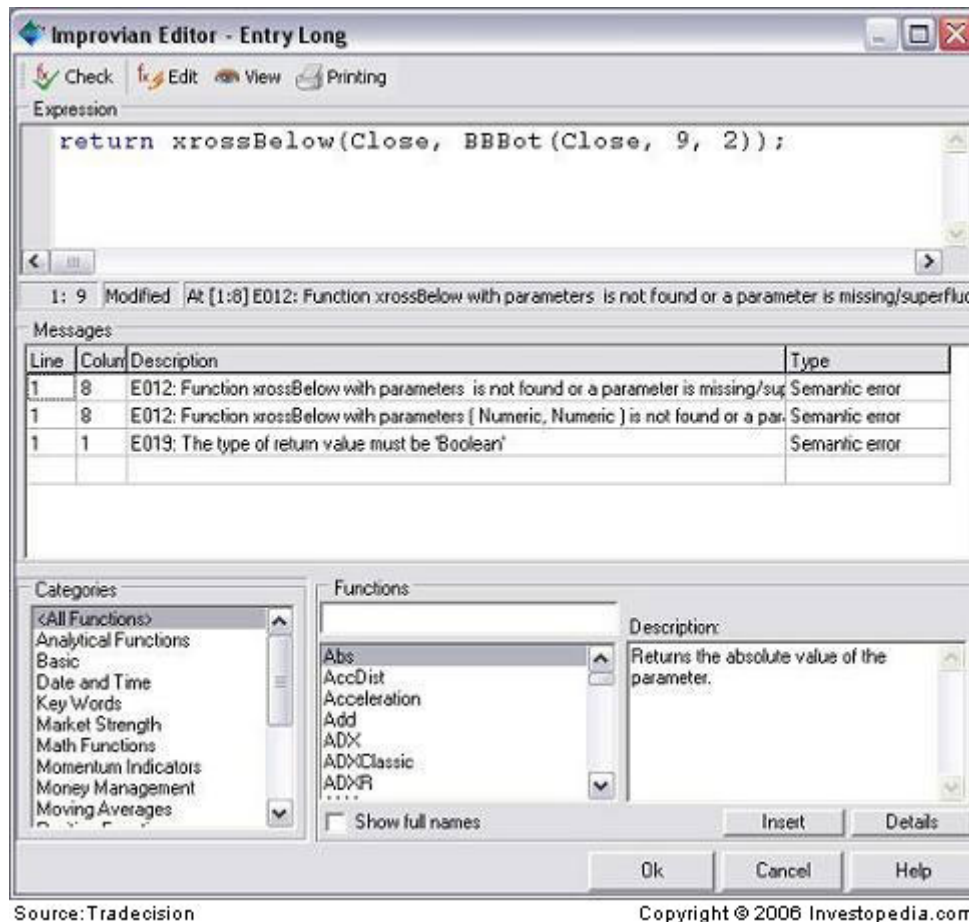
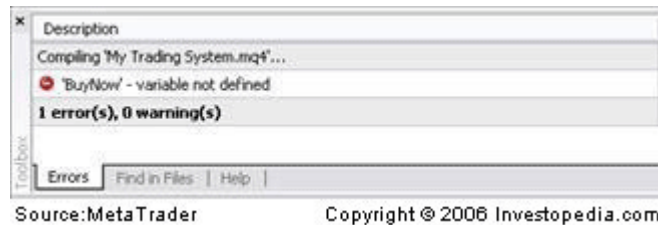


Figure 1

Here we can see that Tradecision gives us the location (line and column) of the error, a description of the error and the type of error (in this case, it is syntactical). If we look at the expression, we can see that in column 8 "xrossBelow" is not a valid function. If we replace the "x" (which is in column 8) with a "c", then we will have valid code.

If we look at MetaTrader, we can see that the errors come up when we try to compile the program:



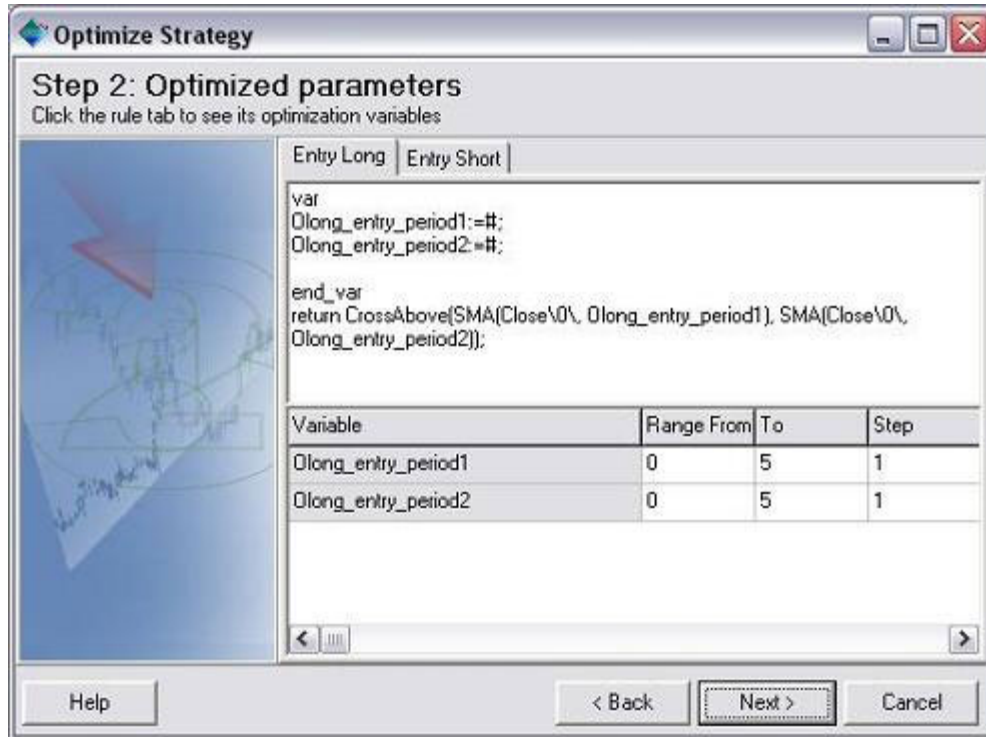
Here we can see that in the description it says the "BuyNow" variable wasn't defined. Double clicking on this error message will bring us to the specific location of the error in the code.

As you can see, most trading applications give you an easy way to locate technical errors and fix them. Fixing the errors simply involves systematically going through each error message and then recompiling the code and/or applying the trading system to your charts.

Optimizing Your Trading System

Some trading applications let you select variables to be optimized. Tradecision, for example, lets you easily select a variable and replace it with code that will attempt optimization. Optimization itself is simply a process that finds the optimal value for a particular trading system element based on past results and performance. Note that over-optimization results in trading systems that are unable to adapt to market conditions; therefore, it is important to only optimize a few important variables, not every variable!

Here is what the optimization feature looks like in Tradecision:



Source:TradeDecision

Copyright © 2006 Investopedia.com

Figure 3

You can see that we declared two new variables and set them equal to "#". The "#" simply means that the trading program will replace this with the optimal number. Next, you can see that we used the new variables within our trading strategy. Finally, we set a range for the numbers (so that the program will not search to infinity).

Some other trading programs have features that operate in a similar way, allowing you to replace the numerical value with a "#" and telling the trading application to optimize it.

Conclusion

By now you should have developed a working trading system in which you can have confidence. In the next part of this series, you will learn how to apply your trading system to charts and how to use it to make trading decisions!

Using Your System

You are now on your way to having a working, profitable trading system. All that is left to do is to apply this trading system to your actual trading. In this section, we will take a look at the ways in which this can be done.

Compiling Your Code

The final step in the actual development of your trading system is compilation - that is, converting your code into a file that the trading software can execute, or run, at any given time without re-reading the code.

The way in which code is compiled differs between trading programs. However, the majority of them simply let you click a compile button and do one of two things: either 1) the program will compile the code and create a new file, or 2) the compiler will list the errors that you have made in your code (as we saw in the previous section). Because MetaTrader has a standard setup, we will use its trading application as an example for the purposes of this tutorial.

MetaTrader's "Compile" button can be found on the top tool bar:



Assuming the compilation goes well, you will now have an executable file that your trading program can quickly read and apply to your charts.

Applying the System to Your Charts

Most trading applications will let you easily apply your trading system within the trading application by either letting you drag the file onto the chart, or inserting it via a menu. MetaTrader allows you to drag the executable file from the "Navigator" window onto the chart to which you wish to apply your trading system.

After this, a dialog box comes up with several options:



Source: MetaTrader

Copyright © 2006 Investopedia.com

Figure 2

Common

The first set of options is standard with many trading applications. The first option simply lets you define what types of positions you are willing to take (long, short, or both). The second option lets you enable "alerts", which are pop-up windows that notify you when your criteria for a trade have been met.

Live Trading

There are two ways in which you can apply your trading system:

1. Semi-Automated Systems - Semi-automated systems are those that alert you to new trades that meet your criteria. Although the alerts themselves are automated, the trades are not placed automatically - hence the "semi" prefix. Although this type of system carries significantly less risk, it also requires you to be near a computer at all times. However, recent innovations have helped solve some of these inconveniences by allowing signals to be sent via email, phone (short message service) or other hi-tech media.

2. Automated Systems - Automated systems are those that place trades with your broker automatically - that is, they require no intervention on your part. This type of trading system involves significantly more risk, especially if there are logical errors that you did not catch when testing. Therefore, it is imperative that you either [paper trade](#) or semi-automate your trading system to be sure that it performs as expected in a live environment. (For further reading, see [Demo Before You Dive In.](#)) Note that these trading systems will also require you to

complete additional paperwork for your broker stating that they can't be held responsible if your trading system generates large losses.

Safety

The two options here (see Figure 2) let you determine whether or not you are willing to let the program call external dynamic link libraries (DLLs). Remember that DLLs are libraries that let you reuse code from other people's trading systems. If your trading system makes use of these external DLLs, then you will need to enable these options. If not, then you are best off leaving these unchecked.

Inputs

Here is where you can define the inputs for the trading system if you did not specify them directly in your code:

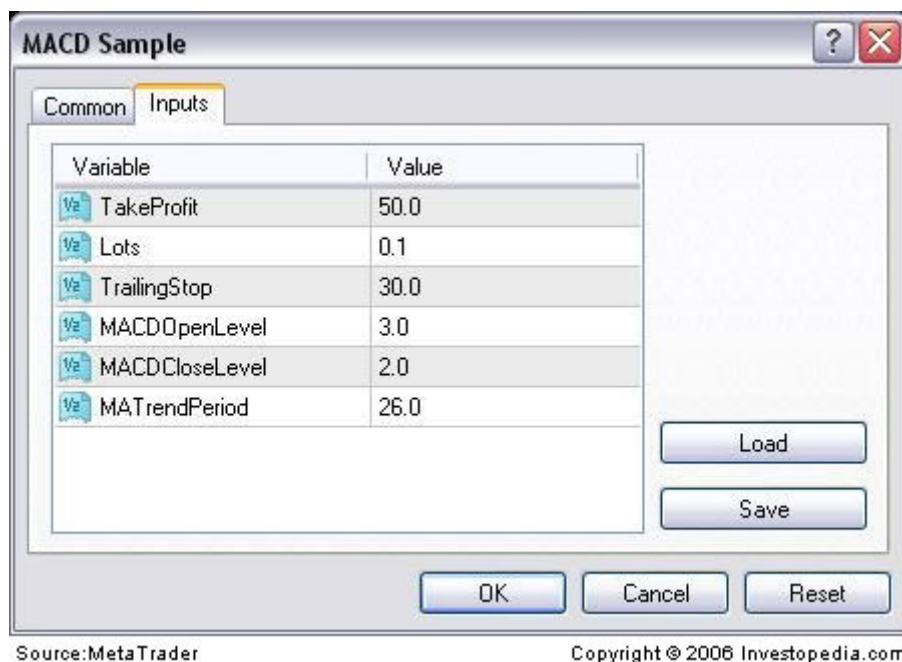


Figure 3

Notice that this area enables you to insert custom inputs without modifying the code at all. This is useful if you plan on changing your inputs, but want to use the same basic strategy. Note that if you optimized your variables, this option would not be available.

Conclusion

Now you should be able to compile and apply your trading system! Again, be sure to paper trade - or at least semi-automate - your trading system before

allowing the system to place trades automatically. Failure to do this could lead to large losses should there be a logical error in your code.

Conclusion

After going through this step-by-step tutorial, you should have a fully operational and fully automatic trading system.

As you continue working with your system, keep in mind the following:

- Always backtest until your system performs well with past data, then paper trade to make sure your system performs well with current data.
- The market has two phases - [trending](#) and [ranging](#) - and very few trading systems handle both perfectly. Be sure to only trade in a market that your system can beat.
- Make changes one at a time so that you can pinpoint which aspects are improving your returns and which are hurting them.
- Keep it simple. Extremely complex trading systems are often fitted to work well with past data, but are incapable of adapting to new market conditions.
- Make sure that you know the strategy behind your trading system. As absurd as it sounds, many people develop their systems until they become so complex that they forget the underlying strategy.
- Don't over-optimize. Optimizing too much can lead to what is known as curve-fitting, which can reduce your trading system's effectiveness and ability to adapt.

Resources

Here are some trading applications worth checking out:

MetaTrader - <http://www.metaquotes.net/>

TradeStation - <http://www.tradestation.com/>

Tradecision - <http://www.tradecision.com/>

MetaStock - <http://www.metastock.com/>

AmiBroker - <http://www.amibroker.com/>

WealthLab - <http://www.wealth-lab.com/>

Comprehensive List - <http://elitetrader.com/so/>

Here are some community resources that can assist you:

Moneytec - <http://www.moneytec.com/>

StrategyBuilderFX - <http://www.strategybuilderfx.com/>

EliteTrader - <http://www.elitetrader.com/>