



4760 E. Fulton, Suite 201 Ada, Michigan USA

phone | 1.616.974.3663
toll free | 1.800.416.7209
fax | 1.616.942.4050

BASIC INTRODUCTION TO CTL FOR CHART STUDIO™

Revised: October 2007

TABLE OF CONTENTS

What is the CTL Programming Language 2

Basic Components of CTL..... 2

 Variables..... 2

 Constants..... 2

 Assignments..... 3

 Statements..... 5

Building an Indicator 6

 MACD Indicator..... 7

Installing an Indicator 7

Additional Information..... 7



WHAT IS THE CTL PROGRAMMING LANGUAGE?

Chart Studio™ enables you to create custom indicators using an easy to understand programming language called the Common Technical Analysis Language (CTL). This programming language is a set of instructions that are used to translate words and symbols into a code that DealBook® FX 2 can understand.

BASIC COMPONENTS OF CTL

Variables

A variable is a symbol into which data can be stored. The value, or contents, of a variable can change during the program's execution.

- A variable assumes a value from a particular range which is called the variable's data type. CTL supports four basic data types: Number, Bool, Series, and String.
- Number - This data type may contain positive or negative numbers. For example: 100, -200, 23.45, -0.456, 99999.
- Bool - This data type, also called a logical variable, may contain only two possible values: True or False.
- Series - This data type is most often used to store prices (Close, High, etc.) and the results of the indicator's execution. It acts like a dynamic set of numbers, meaning the size of any series is automatically adjusted by the program.
- You can add or subtract two series, multiply or divide a series by a number, negate a series, or get an element of a series using an operator.
- String - This data type may contain a sequence of characters. For example, abcd, 34H*, ptr7!F*

In order to use a variable in a program, it must be declared. When you declare a variable, you indicate a data type and name for it.

The first letter of the name must be alphabetic and may be as long as you would like. It is advisable to use names that will help make the program easier to read and follow.

There are two basic formats that can be used for variable declaration:

1. **vars** Name(type); where Name is the unique name of the variable and type is the data type of the variable
For example: vars Var1(number) declares Var1 as a variable with a number data type.
2. **vars** Name = Initializer; where Name is the unique name of the variable and Initializer is a constant (additional information on constants can be found below).
For example: vars Var1 = 0 declares Var1 as a variable whose value is set to zero.

Constants

A constant is a variable whose value cannot be changed. Each constant is associated with a certain data type: Number, Bool, Series, and String.

- Immediate Constants:
 - Number constant – a set of digits
 - Boolean constant – true or false
 - String constant – a set of many characters enclosed within double quotes
- External Constants:
 - An external constant describes the data of the chart to which an indicator is attached.
- Boolean Constants:
 - Long – true, if the position is long; false, if there is no position, or the current position is short
 - Short – true, if the position is short; false, if there is no position, or the current position is long
- String Constants:
 - Version – current version number of CTL
 - Symbol – symbol name of a chart to which an indicator is attached
- Series Constants:
 - Open – series of Open price field, which is the price of the first trade of a period
 - High – series of High price field, which is the highest price that a currency has traded during the period
 - Low – series of Low price field, which is the lowest price that a currency was traded during the period
 - Close – series of Close price field, which was the last price that a currency was traded during the period
 - Volume – series of Volume, which is the number of lots that were traded during the period
 - Openint – series of Open Interest, which is the total number of outstanding lots of a Currency
 - Timestamp – series of timestamps of a time period's start. Timestamp is expressed as a number of seconds elapsed since midnight of January 1, 1970



Assignment

Once you have declared a variable, you can store data in it. This is called an assignment.

The basic format for an assignment is Variable := expression; where the expression can be a variable, a constant, an arithmetic sequence, an indicator, or a combination of these.

- Operator [] - This operator allows you to retrieve an element of a series at a specified position. The position must be a positive integer and must be enclosed in brackets ([]).
- Arithmetic - used to build arithmetic expressions.

Operator	Description	Left Operand	Right Operand	Result
-	Unary negation	-	Number	Number
-	Unary negation (to negate each element of a series)	-	Series	Series
*	Multiplication	Number	Number	Number
*	Multiplication (to multiply each element of a series by a number)	Number	Series	Series
*	Multiplication (to multiply each element of a series by a number)	Series	Number	Series
/	Division	Number	Number	Number
/	Division	Series	Number	Series
+	Addition	Number	Number	Number
+	Addition	Series	Series	Series
+	Addition (string constant)	String	String	String
-	Subtraction	Number	Number	Number
-	Subtraction	Series	Series	Series

- Logical - used to build logical expressions.

Operator	Description	Left Operand	Right Operand	Result
NOT	Logical NOT	-	Bool	Bool
AND	Logical AND	Bool	Bool	Bool
OR	Logical OR	Bool	Bool	Bool



- Relational - used to build relational expressions.

Operator	Description	Left Operand	Right Operand	Result
<	Less than	Number	Number	Bool
>	Greater than	Number	Number	Bool
<=	Less than or equal to	Number	Number	Bool
>=	Greater than or equal to	Number	Number	Bool
<>	Not equal	Number	Number	Bool
<>	Not equal	Number	Number	Bool
=	Equal	Bool	Bool	Bool
=	Equal	Number	Number	Bool

The following table shows the CTL operators' precedence. The operator with the highest precedence is at the top of the table.

Unary +, -
*, /
+, -
NOT
AND
OR
<, >, <=, >=, <>, =

Chart Studio™ works with arithmetic expressions according to the following rules:

1. Look for all expressions in parentheses, starting from the innermost set of parentheses and proceeding to the outmost.
2. Look for all unary minuses and pluses.
3. Look for all multiplication and division from left to right
4. Look for all addition and subtraction from left to right



Statements

Statements allow for using standard algorithmic operations, such as conditions and loops.

- Conditional Statements

Conditional, or control-of-flow, statements are used to determine what code should run in a given situation.

IF...THEN and IF...THEN...ELSE Statements

After the if keyword, is the condition, which is a Boolean type (true or false). The then keyword separates the condition from the action that must be taken if the condition is true.

```
vars Var1 = 1, Var2(series), Var3(series);  
if Var1 < Var2 then return;
```

In this example, the return statement is invoked if the variable Var1 is less than the variable Var2.

If the action is not finished with a semicolon, but the else keyword is present, the action after the else keyword will be taken in case of a false result.

```
if Var2 > Var3 then  
    buy()  
else  
    sell();
```

In this example, if the variable Var2 is greater than variable Var3, then a buy signal is invoked, otherwise, a sell signal is invoked.

In many cases, it is required to take multiple actions if the condition is true or false. In this case, begin...end operation parentheses must be used.

For definitions of internal functions such as buy() and sell(), please see the Chart Studio™ User Guide

- Loop Statements

The ability to iterate through a section of code a prescribed number of times or until a certain condition is reached is called a loop. The CTL programming language offers three kinds of loops: for-to-do, for-downto-do, and while-do.

FOR...TO...DO and FOR...DOWNTO... DO Statements

The **for** keyword is followed by an assignment operator, which sets the initial value for the loop variable. The expression after the to or downto keyword is the last value of this variable. The loop action that follows the do keyword will be executed for each value of this variable from the range (increasing the value by one after every run of the loop action for **to** and decreasing for **downto**).

```
for i := 1 to 5 do  
    i := i + 1;
```

In the example above, the loop will run, each time adding 1 to the variable i equals 5.

```
for i := 10 downto 5 do begin  
    i := i - 1;  
end
```



4760 E. Fulton, Suite 201 Ada, Michigan USA

phone | 1.616.974.3663
toll free | 1.800.416.7209
fax | 1.616.942.4050

In the example above, the loop will run, counting down from 10 to 5, subtracting 1 from i during each iteration.

WHILE...DO Statement

The while keyword is followed by a Boolean condition (true or false). The action follows the do keyword and is executed each time the condition is true.

```
i := 5
while i < 10 do
i := i + 1;
```

In the above example, the loop will run until the variable i is greater than 10.

- Break Statement
This statement causes current for...to...do or while...do statements to terminate without any conditions.
- Continue Statement
This statement causes the current loop to make the next step or terminate (depending on the loop condition).
- Return Statement
This statement causes your module to finish execution and returns the current result value. This statement is usually used to terminate an indicator if the input parameters are not correct or there is not enough data in the series.

BUILDING AN INDICATOR

For each indicator there is a certain mathematical model, generally a set of formulas. The indicator accepts data in input parameters and returns a result as a set of a series.

Every indicator consists of the following parts:

Header:

indicator Name; unique name of the indicator

Input parameters declaration:

input Parameters; list of comma separated parameters of the indicator

Result declaration:

draw Lines; list of comma separated lines of an indicator

Local variables declaration:

vars Variables;

Code:

begin
Statements; a list of semicolon separated statements
end.



MACD Indicator

```
1. indicator MACD;  
2. input src = close,  
3.   first_period = 12,  
4.   second_period = 26,  
5.   signal_period = 9;  
6. draw res("MACD"), signal("MACD sig");  
7. begin  
8. res := ema(src, first_period) - ema(src, second_period);  
9. signal := ema(res, signal_period);  
10. end.
```

Line 1 uses the keyword **indicator** to declare a new indicator with the name MACD.

The declaration of parameters follows on lines 2 through 5 and starts with the keyword **input**. The value of src is set to the predefined constant close, which calculates the value of close prices. The three periods of the MACD indicator are represented by first_period, second_period, and third_period, whose values are numeric data types.

Line 6 declares the lines that must be drawn for the MACD indicator with the keyword **draw**. The MACD consists of two lines which are represented by res (from "result") and signal. Each line has a name that is a series data type, so they are enclosed in double quotation marks.

After the declarations, the execution code follows with the **begin** keyword on line 7.

Since the MACD is the difference between a 26-day and a 12-day exponential moving average, the res line is defined as the result of the subtraction of the exponential moving average (ema) of the source series (src), with period 26 (second_period) from the exponential moving average of src with period 12 (first_period) on line 8.

On line 9, the signal line is defined as the exponential moving average (ema) of res.

Line 10 declares the end of the execution code with the keyword **end**.

INSTALLING AN INDICATOR

Once you have finished building your indicator, you should check for errors by verifying, or compiling, the code. You may do so by clicking on the Build menu and selecting Verify Module. Should any errors be found, they will be listed in the Output Description box located at the bottom of the Chart Studio™ window.

Once you have successfully verified your indicator, you may install it into DealBook® FX by clicking on the Build menu and selecting Install Module. You may then find your indicator in DealBook® FX by right clicking on a chart and selecting the Add Studies option. Your new indicator will be located at the bottom of the list.

ADDITIONAL INFORMATION

You may find additional information about Chart Studio™ and CTL in the Chart Studio™ user guide located at http://www.gftforex.com/documents/manuals/chart_studio.pdf.