

SMSL

Stock Market Scripting Language

Created and Developed

By

Chuck Bolin

Contents

SMEP Scripting Language (SMSL)	4
Scripts	4
Downloading Files	4
Loading File Data into a Worksheet That Does Not Exist	6
Clearing a Worksheet	7
Creating Worksheets	7
Deleting Worksheets	7
Adding Columns	8
Calculating Moving Averages.....	9
Buy and Sell Columns	10
MIN, MAX, and SUM.....	13
Buying and Selling with Results.....	13
Scripting Functions	17
AddCol(arg1, arg2, arg3).....	17
AddSheet(arg1)	17
Clear(arg1)	17
Example: Clear(TestData) Comments.....	18
Define(arg1, arg2, arg3, arg4).....	18
Download(arg1).....	18
Download(arg1, arg2).....	18
Download(arg1, arg2, arg3)	19
EMA(arg1,arg2)	19
Evaluate(arg1).....	19
FillCol(arg1, arg2)	19

FillCol(arg1, arg2, arg3)..... 20
 Arg1 – Historical stock price data file Arg2 – Name of worksheet..... 20
 Example: Load(GOOG.CSV, StockData) 20
 Log(arg1)..... 20
 Arg1 = 0, 1, or 2. 0 = don't save results. 1 = save results. 2 = save results and open 'Notepad.exe'..... 20
 Example: Log(2)..... 20
 Logic(arg1, arg2, arg3) 20
 LogicX(arg1) 21
 Mathematical Expressions..... 21
 Max(arg1, arg2)..... 21
 Min(arg1, arg2) 21
 Setup Functions..... 22
 SMA(arg1,arg2) 23
 Write(arg1, arg2, arg3, arg4) 23

SMEP Scripting Language (SMSL)

The Stock Market Evaluation Program (SMEP) refers to an Excel program written in Visual Basic for Applications. SMEP has an engine which has been designed to process a script file written in the Stock Market Scripting Language (SMSL). SMSL allows a user, via a script, to download historical stock data, to perform numerous calculations, and to evaluate a stock for buy and sell signals. In addition the SMEP can determine the results of investing in a stock over a user-defined period of time.

SMEP is not a 'magic program'. It performs an automatic technical analysis defined by the user. This document does not presume to pick the best or most profitable stocks. It is the responsibility of the user to conduct his or her own fundamental and technical analysis before making buy and sell decisions.

The purpose of SMEP is to automate the process of 'reading charts'.

Historical Stock Data

Historical stock data is available free from the website <http://finance.yahoo.com>. Visitors may download comma-separated value (CSV) files and specify the range of data.

Scripts

Scripts may be written using any text editor. The author has used the "notepad.exe" program for creating the scripts. Scripts are saved with a '.txt' extension in a application subfolder 'scripts'. This document will use examples of scripts as the primary means of teaching the user to create them. In addition, techniques and limitations will be discussed when appropriated.

Downloading Files

```
//*****  
//DOWNLOAD1.TXT - Written by Chuck Bolin, 10/2008  
//Purpose: Demonstrates the method for downloading  
//historical stock files.  
//*****
```

```
//Download last year of Yahoo stock prices.  
//Requires a stock symbol as the primary argument.  
Download(YHOO)
```

```
//Downloads the last 90 days of Google stock prices.  
Download(GOOG, 90)
```

```
//Downloads Amazon stock prices from January 1, 2005  
//to March 1, 2005
```

Download(AMZN, 1/1/2005, 3/1/2005)

The script "Download1.txt" downloads three stocks. One year of the most recent Yahoo (YHOO) price data is downloaded. The last 90 days of Google (GOOG) is downloaded. Stock data between 1/1/2005 and 3/1/2005 is downloaded.

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Volume	Adj Close
2	7/7/2008	542.30	549.00	535.60	543.91	4255200	543.91
3	7/8/2008	545.99	555.19	540.00	554.53	4932400	554.53
4	7/9/2008	550.76	555.68	540.73	541.55	4154000	541.55
5	7/10/2008	545.00	549.50	530.72	540.57	4331700	540.57
6	7/11/2008	536.50	539.50	519.43	533.80	4981400	533.80
7	7/14/2008	539.00	540.06	515.45	521.62	4424800	521.62

Sample Data Displayed in Excel

Historical stock data is saved in the form of 7 comma delimited values. The very first row is called the header and includes the name of each column. The filenames are AMZN.CSV, GOOG.CSV, and YHOO.CSV. The three files are stored in the subfolder "history".

Comments or remarks may be inserted into the script by adding // at the beginning of the comment.

Loading File Data into a Worksheet

```
//*****
//LOAD1.TXT - Written by Chuck Bolin, 10/2008
//Purpose: Demonstrates the method for downloading
//historical stock files and loading it into a
//worksheet.
//*****
```

```
//Downloads the last 100 days of IBM stock prices.
```

Download(IBM, 100)

```
//Loads file IBM.CSV into worksheet "StockData"
```

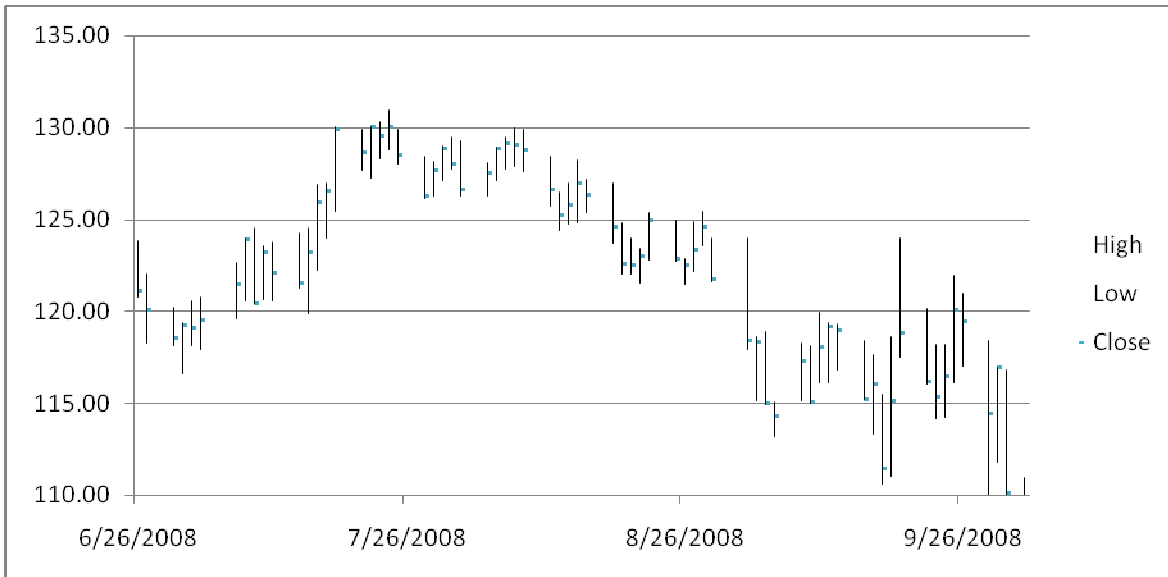
Load(IBM.CSV, StockData)

	A	B	C	D	E	F
1	Date	Open	Volume	High	Low	Close
2	6/26/2008	123.43	9710600	123.82	120.76	121.13
3	6/27/2008	121.02	11660400	122.05	118.26	120.05
4	6/30/2008	120.03	8439000	120.22	118.15	118.53
5	7/1/2008	117.50	10069400	119.36	116.60	119.27
6	7/2/2008	118.41	8093200	120.55	118.12	119.10

Loaded IBM Data

Comparing the loaded data format to the downloaded data format, it can be seen that one column of data is missing. "Adj Close" data is not loaded since it does not contribute the evaluation of stock data. Additionally, the order of data has been changed.

The data is downloaded and loaded such that the oldest data is at the top of the worksheet. The most recent data is at the bottom of the worksheet. Below is a sample chart in the High-Low-Close chart style.



Sample Data Displayed in a Chart

Loading File Data into a Worksheet That Does Not Exist

This script creates a worksheet required for the downloaded data.

```
//*****
//LOAD2.TXT - Written by Chuck Bolin, 10/2008
//Purpose: Demonstrates the method for downloading
//historical stock files and loading it into a
```

```
//worksheet.
//*****
```

```
//Downloads the last 100 days of IBM stock prices.
```

```
Download(IBM, 100)
```

```
//Loads file IBM.CSV into worksheet "IBMDData"
```

```
Load(IBM.CSV, IBMDData)
```

If it is necessary to run the script 'Load2.txt' more than once, it is acceptable to 'comment out' the line that reads "Download(IBM, 100)". Add two forward slashes in front of this line so it looks like:

```
//Download(IBM, 100)
```

Clearing a Worksheet

```
//*****
//CLEAR.TXT - Written by Chuck Bolin, 10/2008
//Purpose: Demonstrates clearing worksheet data.
//*****
```

```
Clear(StockData)
```

Creating Worksheets

```
//*****
//ADDSHEET.TXT - Written by Chuck Bolin, 10/2008
//Purpose: Creates several worksheets
//*****
```

```
AddSheet(IBM)
```

```
AddSheet(GOOG)
```

```
AddSheet(YHOO)
```

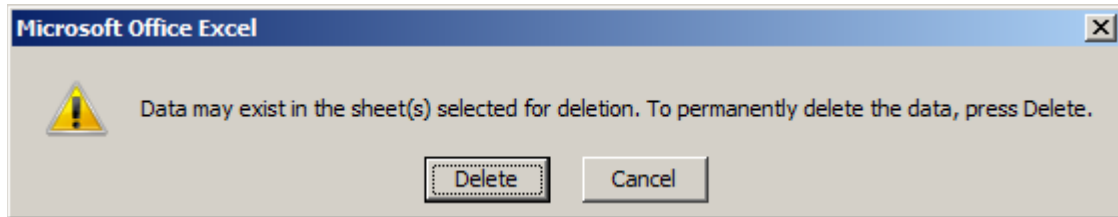
```
AddSheet(AMZN)
```

Deleting Worksheets

```
//*****
//DELETE.TXT - Written by Chuck Bolin, 10/2008
//Purpose: Deletes multiple worksheets
//*****
```

Delete(IBM)
Delete(GOOG)
Delete(YHOO)
Delete(AMZN)

For each delete function in the script above, a message box is displayed below.



Delete Worksheet Warning

Adding Columns

Adding columns is a significant and important detail to SMEP. Failure to do this will cause all sorts of scripting problems. Calculations are performed along the entire range of the stock price data within the worksheet. The name of the columns is instrumental in making the SMEP Engine work properly.

```
//*****
//SAMPLE1.TXT - Written by Chuck Bolin, 10/2008
//Purpose: Adds columns
//*****
```

```
//downloads Game Stop (GME) data
```

Download(GME)

```
//clears worksheet 'StockData'
```

Clear(StockData)

```
//Loads stock data into worksheet
```

Load(GME.CSV, StockData)

```
//Adds column - seems redundant but is necessary
```

```
//Three arguments: name of worksheet, name of
```

```
//column (placed at the top), and column number
```

AddCol(StockData, Date, 1)

AddCol(StockData, Open, 2)

AddCol(StockData, Volume, 3)

AddCol(StockData, High, 4)

AddCol(StockData, Low, 5)

AddCol(StockData, Close, 6)

```
//Adds some columns to store calculations
```

```
AddCol(StockData, HMA6, 7)
```

```
AddCol(StockData, LMA6, 8)
```

	A	B	C	D	E	F	G	H
1	Date	Open	Volume	High	Low	Close	HMA6	LMA6
2	10/5/2007	55.10	2332400	56.77	54.80	56.72		
3	10/8/2007	56.85	1172200	57.90	56.30	56.65		

Additional Columns

It should be noted that column labels are added each time historical stock data is loaded into the worksheet. However, this is not sufficient. It is necessary to add the columns like in the example above. The explanation is that it is possible to load multiple stocks and index data into worksheets in the same script. A name clash could easily occur since each file would have a Date, Open, Volume, etc. The user must assign unique names if multiple files are loaded.

Calculating Moving Averages

The scripting function 'SMA' is a Simple Moving Average. It calculates the average for a specific number of values within a range. The following sample performs two calculations and places them into the columns named HMA6 and LMA6.

```
//*****
//SAMPLE2.TXT - Written by Chuck Bolin, 10/2008
//Purpose: Moving Average calculations.
//*****
```

```
//downloads Game Stop (GME) data
```

```
Download(GME)
```

```
//clears worksheet 'StockData'
```

```
Clear(StockData)
```

```
//Loads stock data into worksheet
```

```
Load(GME.CSV, StockData)
```

```
//Adds column - seems redundant but is necessary
```

```
//Three arguments: name of worksheet, name of
```

```
//column (placed at the top), and column number
```

```
AddCol(StockData, Date, 1)
```

```
AddCol(StockData, Open, 2)
```

```
AddCol(StockData, Volume, 3)
```

```
AddCol(StockData, High, 4)
```

```
AddCol(StockData, Low, 5)
```

```
AddCol(StockData, Close, 6)
```

```
//Adds some columns to store calculations
```

```
AddCol(StockData, HMA6, 7)
```

```
AddCol(StockData, LMA6, 8)
```

```
//Performs simple moving average calculations
```

```
HMA6 = SMA(High, 6)
```

```
LMA6 = SMA(Low, 6)
```

Buy and Sell Columns

```
//*****
```

```
//SAMPLE3.TXT - Written by Chuck Bolin, 10/2008
```

```
//Purpose: Buying and selling
```

```
//*****
```

```
//downloads Game Stop (GME) data
```

```
Download(GME)
```

```
//clears worksheet 'StockData'
```

```
Clear(StockData)
```

```
//Loads stock data into worksheet
```

```
Load(GME.CSV, StockData)
```

```
//Adds column - seems redundant but is necessary
```

```
//Three arguments: name of worksheet, name of
```

```
//column (placed at the top), and column number
```

```
AddCol(StockData, Date, 1)
```

```
AddCol(StockData, Open, 2)
```

```
AddCol(StockData, Volume, 3)
```

```
AddCol(StockData, High, 4)
```

```
AddCol(StockData, Low, 5)
```

```
AddCol(StockData, Close, 6)
```

```
//Add Buy and sell columns
```

```
AddCol(StockData, Buy, 7)
```

```
AddCol(StockData, Sell, 8)
```

```
//Pre-fill columns with 1's
```

```
FillCol(Buy, 1)
```

```
FillCol(Sell, 1)
```

```
//Adds some columns to store calculations  
AddCol(StockData, HMA6, 9)  
AddCol(StockData, LMA6, 10)
```

```
//Performs MA calculations  
HMA6 = SMA(High, 6)  
LMA6 = SMA(Low, 6)
```

```
//Determines buy and sell  
Buy = LogicX(Close > HMA6)  
Sell = LogicX(Close < LMA6)
```

The Sample3.txt scripts downloads stock data, loads the data into the worksheet, adds column names, fills two columns with 1's, performs two simple moving average calculations, and determines buy and sell. The triggers to buy and sell are indicated by a '1' in the column. The data along with the Buy and Sell signals are shown below.

The moving average information does not exist above the black solid line. The Buy and Sell signals are ignored above this line. The Buy signal instructs the investor to buy GME at \$60.05 per share on 10/23/2007. The Sell signal instructs the investor to sell GME at \$56.33 per share on 11/2/2007.

Obviously the rules for Buy and Sell have not been correctly determined. However the example serves the purpose of illustrating the ability of SMEP. The user (investor) of SMEP will spend over 90% of his or her time adjusting the buy and sell rules.

Date	Open	Volume	High	Low	Close	Buy	Sell	HMA6	LMA6
10/8/2007	56.85	1172200	57.90	56.30	56.65	1	0		
10/9/2007	57.21	2855200	58.89	57.08	58.71	1	0		
10/10/2007	58.75	2179700	59.45	58.00	58.97	1	0		
10/11/2007	59.13	1557400	59.40	56.00	57.05	1	0		
10/12/2007	56.93	2008300	57.49	56.03	57.40	1	0		
10/15/2007	58.10	2779600	59.49	57.28	58.02	0	0	58.77	56.78
10/16/2007	57.49	1906000	59.2	57.49	58.12	0	0	58.99	56.98
10/17/2007	59.48	3129500	60.15	57.72	58.51	0	0	59.20	57.09
10/18/2007	58.4	1304700	59.13	57.82	58.39	0	0	59.14	57.06
10/19/2007	58.51	1249100	58.97	56.7	56.74	0	1	59.07	57.17
10/22/2007	55.9	2590500	58	53.25	56.76	0	0	59.16	56.71
10/23/2007	57.22	2453800	60.13	56.65	60.05	1	0	59.2633	56.61 BUY
10/24/2007	60.65	1851000	60.8	56.96	58.13	0	0	59.53	56.52
10/25/2007	58.41	2115300	58.68	55.85	56.61	0	0	59.285	56.21
10/26/2007	58.57	3294300	58.95	57	57.44	0	0	59.255	56.07
10/29/2007	58.84	2482800	59.46	57.96	58.27	0	0	59.3367	56.28
10/30/2007	58.1	1176000	59	57.48	57.5	0	0	59.5033	56.98
10/31/2007	57.81	2456000	59.25	57.73	59.22	0	0	59.3567	57.16
11/1/2007	58.75	2255300	60.5	57.85	57.85	0	0	59.3067	57.31
11/2/2007	58.95	2189400	59.45	55.36	56.53	0	1	59.435	57.23 SELL

Buy and Sell Signals

The function 'LogicX' requires some explanation. If the logical expression within the parenthesis is 'true' then the destination column value is multiplied by '1'. If the logical expression is 'false' then the destination column is multiplied by '1'.

Mathematical Operations and Complex Logical Expressions

The following script snippet defines three more calculations. HCDIFF and CLDIFF are simple math expressions.

```
//Adds some columns to store calculations
AddCol(StockData, VMA10, 11)
AddCol(StockData, HCDIFF, 12)
AddCol(StockData, CLDIFF, 13)

//Performs MA calculations
VMA10 = SMA(Volume, 10)
HCDIFF = Close - HMA6
CLDIFF = Close - LMA6
```

```
//Determines buy and sell
```

```
Sell= LogicX(Close > HMA6 & Volume > VMA10 & Close < Close[-2])
```

```
Buy = LogicX(Close < LMA6 & Volume > VMA10 & Close > Close[-2])
```

The 'LogicX' function above each includes three expressions separated by the 'and' symbol '&'. For the Sell calculation, it can be stated as follows.

Sell if **Close > HMA6** and **Volume > VMA10** and **Close < Close[-2]**

The term 'Close[-2]' deserves an explanation. The bracketed -2 refers to the current row minus two rows. So 'Close<Close[-2]' is true if the current Close price is less than the price two days earlier.

MIN, MAX, and SUM

The scripting language includes support for determining the minimum, maximum, and sum of a range of numbers.

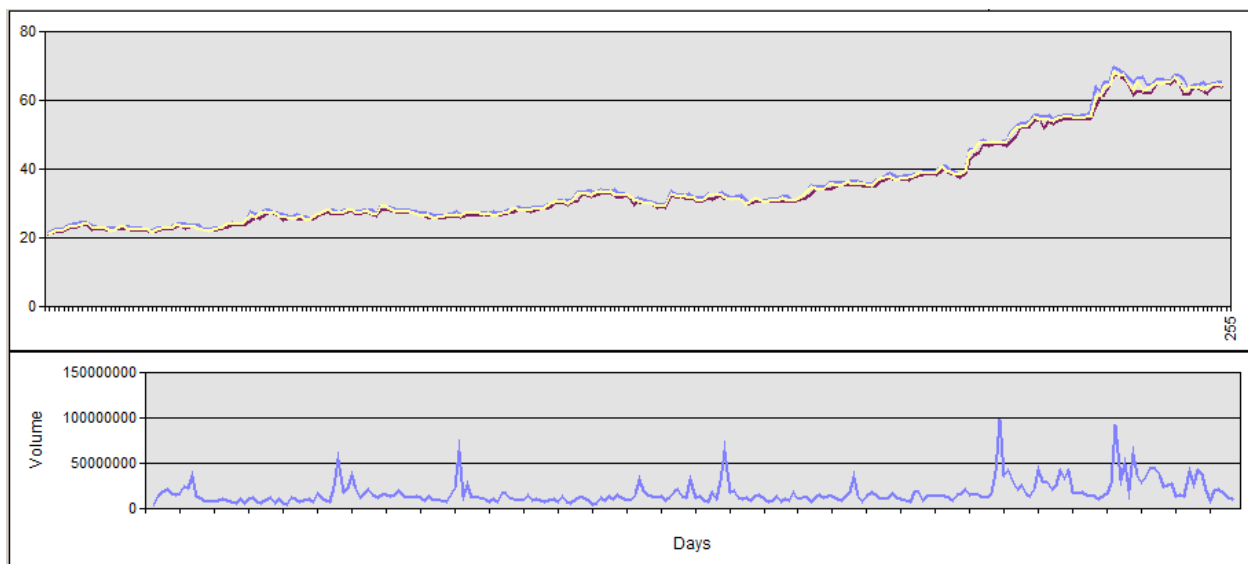
```
CMA6 = MAX(Close, 6)
```

```
CMIN6 = MIN(Close, 6)
```

```
CSUM6 = SUM(Close, 6)
```

Buying and Selling with Results

The following program downloads 'Apple (AAPL)' stock during the period of 01/01/2004 and 01/01/2005. The data is loaded into a worksheet, calculations are performed, and the buying and selling signals are determined. Results are displayed. The chart below indicates that the closing price is on a steady rise.



```
//*****
//BULL3.TXT - Written by Chuck Bolin, 10/2008
//Purpose: Buying and selling
//*****

#include base1.txt

//downloads data
Download(AAPL, 01/01/04, 01/01/05) //apple

//clears worksheet 'StockData'
Clear(StockData)

//Loads stock data into worksheet
Load(AAPL.CSV, StockData)
SetStart(03/01/04) //this is the earliest that buying of stock may occur

//Add columns
AddCol(StockData, Date, 1)
AddCol(StockData, Open, 2)
AddCol(StockData, Volume, 3)
AddCol(StockData, High, 4)
AddCol(StockData, Low, 5)
AddCol(StockData, Close, 6)
AddCol(StockData, Buy, 7)
AddCol(StockData, Sell, 8)
AddCol(StockData, Comment, 9)

//Pre-fill columns with 1's
FillCol(Buy, 1)
FillCol(Sell, 1)

//Adds some columns to store calculations
AddCol(StockData, HMA13, 10)
AddCol(StockData, LMA6, 11)
AddCol(StockData, HMA6, 12)
AddCol(StockData, PMA10, 13)
AddCol(StockData, EMA26, 14)
AddCol(StockData, ACC20, 15)
AddCol(StockData, DIST20, 16)
AddCol(StockData, ACCUM, 17)
AddCol(StockData, DISTRIB, 18)
AddCol(StockData, A_D, 19)

//Performs MA calculations
HMA13 = SMA(High, 13)
LMA6 = SMA(Low, 6)
```

```

HMA6 = SMA(High, 6)
PMA10 = SMA(Close, 10)
EMA26 = EMA(Close, 26)
ACC20 = Logic(Close > Close[-1] & Volume > Volume[-1], 1, 0)
DIST20 = Logic(Close < Close[-1] & Volume > Volume[-1], 1, 0)
ACCUM = SUM(ACC20, 20)
DISTRIB = SUM(DIST20, 20)
A_D = ACCUM - DISTRIB

```

```
//Buy and sell rules
```

```

Buy = LogicX(LMA6 > EMA26 & HMA6 > HMA13 & HMA6 > PMA10 & Close > EMA26 & A_D > 3)
Buy = LogicX(Volume > 10000000)
Sell = LogicX(HMA6 < HMA13)

```

```
Evaluate(1)
```

```
Log(2)
```

The above program produces the following results. An initial investment of \$1000 results in a total value of \$3,729.43 or an overall gain of 272%.

```
*****
```

```
SMEP (Stock Market Evaluation Program) v0.06
```

```
*****
```

SUMMARY OF RESULTS

```

Stock Symbol:      AAPL
Number of shares held: 0
Current share price: 64.4
Cash Value:        3729.43
Total value:      3729.43
Gain/Loss:        272.943%

```

```

Buy Limit: 3
Buy Hold Min: 5
Buy Hold Max: 255
Buy Price: 21.55
Sell Price: 75.1
Sell Gain: 6
Sell Loss: -6
Fee: 7
Init Amount: 1000
EvalStartDate: 3/1/2004

```

A log of buying and selling is displayed below.

```

3/4/2004: Buy: 79 sh
3/5/2004: Sold(4): 79 sh  Gain: 6.28

```

3/8/2004: Buy: 81 sh
3/10/2004: Sold(4): 81 sh Gain: 6.46

3/11/2004: Buy: 82 sh
3/22/2004: Sold(1): 82 sh Loss: -4.75%

3/30/2004: Buy: 75 sh
4/26/2004: Sold(1): 75 sh Loss: -2.83%

6/4/2004: Buy: 71 sh
6/10/2004: Sold(4): 71 sh Gain: 6.81

6/15/2004: Buy: 70 sh
6/16/2004: Sold(4): 70 sh Gain: 6.68

6/17/2004: Buy: 70 sh
7/2/2004: Sold(1): 70 sh Loss: -5.27%

8/31/2004: Buy: 64 sh
9/17/2004: Sold(4): 64 sh Gain: 7.68

9/21/2004: Buy: 62 sh
10/6/2004: Sold(4): 62 sh Gain: 6.92

10/7/2004: Buy: 63 sh
10/14/2004: Sold(4): 63 sh Gain: 13.53

10/15/2004: Buy: 61 sh
10/27/2004: Sold(4): 61 sh Gain: 10.55

10/28/2004: Buy: 60 sh
11/12/2004: Sold(4): 60 sh Gain: 6.34

11/15/2004: Buy: 60 sh
11/22/2004: Sold(4): 60 sh Gain: 11.06

11/23/2004: Buy: 60 sh
11/29/2004: Sold(4): 60 sh Gain: 11.7

11/30/2004: Buy: 61 sh
12/3/2004: Sold(4): 61 sh Loss: -6.52

12/17/2004: Buy: 59 sh
12/27/2004: Sold(1): 59 sh Loss: -2.82%

Scripting Functions

AccDist(arg1)

Arg1 = column name storing immediate results.

The AccDist function calculates accumulate and distribution. If the closing price is higher than yesterday with an increase in volume, the value stored in column arg1 is 1. If the closing price is lower than yesterday with an increase in volume, the value stored in column arg1 is -1. A 0 is stored if volume is down.

The destination column stores the SUM of the last 20 days of values stored in arg1.

Example: ADSUM = AccDist(A_D) //1,0,-1 stored in 'A_D'. Sum of 20 days is stored in 'ADSUM'

AddCol(arg1, arg2, arg3)

Arg1 = worksheet name

Arg2 = name of header (top row)

Arg3 = column number to add header label

This function places in name in the top row of the specified column and worksheet.

Example: AddCol(StockData, HMA6, 7)

AddSheet(arg1)

Arg1 = worksheet name

Example: AddSheet(IBM) //Adds a worksheet and changes the name to 'IBM'.

Clear(arg1)

Arg1 = worksheet name

This function clears the entire contents of the specified worksheet.

Example: Clear(TestData)

Comments

Single lines may be commented out so that they are ignored by two forward slash characters `“//”`. Multiple lines may be commented by beginning with a `“/*”` and ending with a `“*/”`.

Define(arg1, arg2, arg3, arg4)

Arg1 = variable name

Arg2 = worksheet name

Arg3 = column number

Arg4 = row number

This function assigns a variable name to a specific cell in a worksheet.

Example: Define(Symbol, Main, 6, 4)

Delete(arg1)

Arg1 = worksheet name

This function deletes the specified worksheet. The sheet named “Main” cannot be deleted.

Example: Clear(TestData)

Download(arg1)

arg1 = stock symbol

This function downloads the last 12 months of historical stock data from finance.yahoo.com.

Example: Download(GOOG)

Download(arg1, arg2)

arg1 = stock symbol

arg2 = number of days prior to the current date

This function downloads the last X number of days of historical stock data from finance.yahoo.com.

Example: Download(GOOG, 120)

Download(arg1, arg2, arg3)

arg1 = stock symbol

arg2 = begin date of range

arg3 = end date of range

This function allows the user to specify a range of dates to download historical stock data from finance.yahoo.com.

Example: Download(AMZN, 1/1/2005, 1/1/2007)

EMA(arg1, arg2)

Arg1 = Name of column

Arg2 = Number of samples to include in average calculation.

This function performs an Exponential Moving Average calculation. An EMA weights the last value (most recent value) more than the other values.

Example: HMA6 = EMA(High, 6)

Evaluate(arg1)

Arg1 = 1 or 0. A '1' instructs the engine to perform a Buy/Sell evaluation.

Setting Evaluate(1) requires three columns: Buy, Sell, and Comments. Evaluate is turned off by default.

Example: Evaluate(1)

FillCol(arg1, arg2)

Arg1 = Column name

Arg2 = Value

This function fills a column with a specific numeric value. It fills to the maximum row based upon the data located in column A. This is usually the "Date" column.

Example: FillCol(Buy, 1) //fills column 'Buy' with 1

FillCol(arg1, arg2, arg3)

Arg1 = Column name

Arg2 = Value

Arg3 = Max row number, begins at row 2, row 1 is reserved for the column header

This function fills a column with a specific numeric value.

Example: FillCol(Buy, 1, 200) //fills 200 rows with a 1

#Include filename

The include command is a preprocessor command. It allows scripting functions from other files to be inserted into the main script before it is parsed and processed (interpreted).

Example: #include base.txt

Load(arg1, arg2)

Arg1 – Historical stock price data file

Arg2 – Name of worksheet

Example: Load(GOOG.CSV, StockData)

Log(arg1)

Arg1 = 0, 1, or 2. 0 = don't save results. 1 = save results. 2 = save results and open 'Notepad.exe'.

Example: Log(2)

Logic(arg1, arg2, arg3)

Arg1 – This is a logical condition.

Arg2 – Destination receives this calculation if the condition is true.

Arg3 – Destination receives this calculation if this condition if false.

Example: OBV = Logic(Close>Close[-1] & Volume > Volume[-1], OBV[-1] + Volume, OBV[-1] – Volume)

LogicX(arg1)

Arg1 = Logical expression

This function evaluates the logical expression. If the expression is true then the destination column is multiplied by 1. If the expression is false then the destination column is multiplied by 0. NOTE: Terms used in comparison must be on the same sheet. Use "A = B" format to bring data from one sheet to the sheet requiring the comparison. Future version may change this.

Example: Buy = LogicX(Close > Close[-1])

Mathematical Expressions

The scripting language will process a variety of mathematical expressions. NOTE: Avoid using "R", "C", and "RC" for column labels/names/headers. This will cause potential bugs in the program.

Example: Calc = Close * 1.1 + 200

Example: Results = (Close - Close[-2] + High + Low)/4

Max(arg1, arg2)

Arg1 = Name of column

Arg2 = number of samples

The maximum value from a series of 'n' numbers is returned to the destination cell.

Example: M = Max(Close, 3)

Min(arg1, arg2)

Arg1 = Name of column

Arg2 = number of samples

The minimum value from a series of 'n' numbers is returned to the destination cell.

Example: M = Min(Close, 3)

ProcessList(arg1, arg2, arg3)

Arg1 = File with stock list.

Arg2 = Begin Date.

Arg3 = End Date.

This function allows the user to download and process 20 stocks. Symbols are placed in a text file on separate lines. Set 'Log(1)'. 'Log(2)' will require the user to respond to each log report. Do not use 'Download(...)' or 'Load(...)'.
Example: ProcessList(list1.dat, 01/01/08, 10/01/08)

ProcessList(arg1, arg2)

Arg1 = File with stock list.

Arg2 = Number of days prior to current date.

Example: ProcessList(list1.dat, 90)

ProcessList(arg1)

Arg1 = File with stock list.

Downloads and evaluates 1 year of data prior to the current date.

Example: ProcessList(list1.dat)

Save(arg1, arg2, arg3)

Arg1 = Name of worksheet to save.

Arg2 = New name of worksheet (generally the same as arg1).

Arg3 = Filename of new workbook with copy of sheet.

Example: Save(StockData, Data, Google.xls)

Setup Functions

These functions are loaded to support calculations. The values following the functions are the default values.

SetBuyLimit(3)	//1000	- limits the number of times a buy occurs for the stock
SetBuyHoldMin(5)	//0	- defines the number of days to hold as a minimum
SetBuyHoldMax(255)	//0	-defines the maximum number of days to hold stock
SetBuyPrice(21.55)	//10000.00	-price to buy automatically
SetSellPrice(75.10)	//0.00	-price to sell automatically
SetSellGain(6.0)	//500%	-sell if gain is greater than this value
SetSellLoss(-6.0)	//-500%	-sell if loss is greater than this value
SetFee(7.00)	//10.00	-fee for each transaction (buy, sell)
SetInitAmount(1000)	//2000	- initial balance of cash to begin investing

SMA(arg1,arg2)

Arg1 = Name of column

Arg2 = Number of samples to include in average calculation.

This function performs a Simple Moving Average calculation.

Example: HMA6 = SMA(High, 6)

Sum(arg1, arg2)

Arg1 = Name of column.

Arg2 =number of samples

The sum of all values in a series of numbers is returned to the destination cell.

Example: S = Sum(Close, 3)

Write(arg1, arg2, arg3, arg4)

Arg1 = Worksheet name

Arg2 = Column number

Arg3 = Row number

Arg4 = Value to write

This function allows the user to write values (numbers or text) to a cell on a worksheet.

Example: Write(Main, 6, 3, Stock)

Example: Write(Main, 6, 3, 35.55)