Fortex 6 AlgoX User Guide(Java Script)

Version 1.0

Fortex

Table of Contents

Fortex 6 AlgoX User Guide(Java Script)..... 1
Version 1.0.....1
Table of Contents..... 2
1. Global Functions.....5
1.1 getCurrentAccount..... 5
1.2 getQuote..... 5
1.3 setDefaultExpiry.....6
1.4 getDefaultExpiry..... 7
1.5 getOldestBarIndex.....7
1.6 getCurrentBarIndex.....8
1.7 setDefaultQuantity.....8
1.8 getDefaultQuantity.....9
1.9 getValue..... 9
1.10 isHistoryBar..... 11
1.11 getCurSymbol.....11
1.12 sleep..... 12
1.13 getBidSize..... 12
1.14 getAskSize..... 13
1.15 getCurTime.....13
1.16 getQuoteInfo..... 14
1.17 getTick.....14
1.18 getBarState..... 15
1.19 getSymbol.....16
1.20 BarAsk.....16
1.21 BarBid.....17
1.22 getBarCount.....17
1.23 getpoint..... 18
2. Chart data processing.....19
2.1 Price Functions.....19
2.1.1 high.....19
2.1.2 low..... 20
2.1.3 close.....20
2.1.4 open..... 21
2.1.5 highest..... 22
2.1.6 hhv.....23
2.1.7 llv.....23
2.1.8 lowest.....24

2.2 Time Functions **25**
2.2.1 year 25
2.2.2 month 26
2.2.3 day 26
2.2.4 hour 27
2.2.5 minute 28
2.2.6 second 29
2.2.7 getYear 30
2.2.8 getMonth 31
2.2.9 getDay 32
2.2.10 getMinute 32
2.2.11 getSecond 33
3. Built-In Study Functions **34**
3.1 Average True Range 34
3.2 Bollinger Bands 35
3.3 Commodity Channel Index 37
3.4 Moving Average 38
3.5 MACD 39
3.6 Rate of Change 40
4. Trading Functions **41**
4.1 buyMarket 41
4.2 buyLimit 43
4.3 buyStop 44
4.4 buyThreshold 45
4.5 buy 47
4.6 sellMarket 48
4.7 sellLimit 49
4.8 sellStop 51
4.9 sellThreshold 52
4.10 sell 53
4.11 sendOrder 54
5. Account and Portfolio Functions **56**
5.1 Account Object **56**
5.1.1 getAccountName 57
5.1.2 getValue 57
5.1.3 getEquity 58
5.1.4 getBuyingPower 58
5.1.5 getLeverage_Ratio 59
5.1.6 getRequiredMargin 60
5.1.7 getMaintenanceMargin 60
5.1.8 getLiquidationMargin 61
5.1.9 getMargin_Ratio 61
5.1.10 getCommission 62
5.1.11 getPosition 62

5.1.12 getOpenPL....63
5.1.13 getClosePL....64
5.1.14 getPL....64
5.1.15 getQty....65
5.1.16 getTickets....65
5.1.17 getAvailableMargin....66
5.1.18 getAssetBySymbolName....66
5.2 Asset Object....67
5.2.1 getSymbol....68
5.2.2 getSide....68
5.2.3 getPosition....69
5.2.4 getPrice....69
5.2.5 getOpenPL....70
5.2.6 getClosePL....70
5.2.7 getPL....71
5.2.8 closePosition....72
6. Order Manager Functions....72
6.1 Order Manager Object....72
6.1.1 OrderManager.getOpenOrders....74
6.1.2 OrderManager.getFilledOrders....75
6.1.3 OrderManager.getCancelledOrders....75
6.1.4 OrderManager.cancelOrder....76
6.1.5 OrderManager.replaceOrder....77
6.2 Order Object....78
6.2.1 Order.getSymbol....79
6.2.2 Order.getQty....80
6.2.3 Order.getFilledPrice....80
6.2.4 Order.getStopPrice....81
6.2.5 Order.getLimitPrice....81
6.2.6 Order.getType....82
6.2.7 Order.getMarketable....82
6.2.8 Order.getStatus....83
6.2.9 Order.getRoute....83
6.2.10 Order.getSide....84
6.2.11 Order.getTradeDate....84
6.2.12 Order.getTradeTime....85
6.2.13 Order.getStop_Loss....85
6.2.14 Order.getTake_Profit....86
6.2.15 Order.cancel....86
7. Utility Functions....87
7.1 FunctionParameter Object....87
7.2 call....89
7.3 internalCall....90
7.4 ref....91

8. Quote Object.....91
8.1 Quote info.....91
9. Appendix.....92
9.1 Available Colors.....92

1. Global Functions

1.1 getCurrentAccount

Description:

- **getCurrentAccount ()**

Retrieve the current account object.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objAcc = getCurrentAccount ();
    if (objAcc!= null){
        var accName = objAcc.getAccountName();
        debugPrintln("account:" + accName);
    }
}
function OnTick() {
}
```

1.2 getQuote

Description:

- **getQuote** (*sQuoteType*)

Retrieve current bid or ask price.

Parameters：

1. *sQuoteType*: Required. A String value. The text string constants representing the price type. The following constants are available:

Constant	Comment
Bid	Bid price
Ask	Ask price
last	The last transaction price

Note：

If the return value is 0 or a negative integer value, it means the function failed to retrieve the quote.

Example

JS：

```
function OnInit() {
    }
function OnTick() {
      debugPrintln (getQuote ("Bid"));
}
```

1.3 setDefaultExpiry

Description:

- **setDefaultExpiry (*sExpiry*)**

Set the default value of Time-In-Force (TIF) setting for the order.

Parameters:

2. *sExpiry:* Required. Expiration for the order. The following constants are available:

Constant	Comment
FOK	TIF Setting is “FOK”
GTC	TIF Setting is “GTC”
IOC	TIF Setting is “IOC”

Example

JS:

```
function OnInit() {
}
```

```
function OnTick() {
setDefaultExpiry("GTC");
var Expiry = getDefaultExpiry();
        debugPrintln("Expiry:"+Expiry);
}
```

1.4 getDefaultExpiry

Description:

- **getDefaultExpiry ()**

Retrieve the EXPIRY value.

Parameters:

None, function takes no parameters.

Example

JS:

```
function OnInit() {
}
function OnTick() {
var Expiry = getDefaultExpiry ();
debugPrintln ("Expiry:" + Expiry);
}
```

1.5 getOldestBarIndex

Description:

- **getOldestBarIndex ()**

Returns the bar index of the oldest bar in the series.
This will always return a negative integer value unless no data exists for the symbol in the chart window, in which case it will return null.

Parameters:

None, function takes no parameters.

Example

JS:

```
function OnInit() {
}
function OnTick() {
var bar = getOldestBarIndex ();
debugPrintln ("OldestBarIndex:" + bar);
}
```

1.6 getCurrentBarIndex

Description:

- **getCurrentBarIndex ()**

Returns the current offset into the price series that is loaded in the chart.

Parameters:

None, function takes no parameters.

Example

JS:

```
function OnInit() {
    }
function OnTick() {
var bar = getCurrentBarIndex();
debugPrintln ("CurrentBarIndex:" + bar);
}
```

1.7 setDefaultQuantity

Description:

- **setDefaultQuantity (***iQuantity***)**

Set the default QUANTITY value of the order.

Parameters:

3. *iQuantity*: Required. A positive integer value.

Example

JS:

```
function OnInit() {

setDefaultQuantity (14000);

debugPrintln (getDefaultQuantity());
}
function OnTick() {
}
```

1.8 getDefaultQuantity

Description:

- **getDefaultQuantity ()**

Retrieve the default QUANTITY value of the order.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
debugPrintln (getDefaultQuantity());
}
function OnTick() {
}
```

1.9 getValue

Description:

- **getValue (***sBarType*, *iBarIndex* [,*iNumBars*])

Returns the value of the specific price of the specific bar index relative to the bar currently being processed.

Parameters:

4. *sBarType*: Required. A String value. The following constants are available:

Constant	Comment
open	The open price of specified Bar.
high	The high price of specified Bar.
low	The low price of specified Bar.
close	The close price of specified Bar.
average	The average price of specified Bar.
volume	The volume value of specified Bar.
year	The year value of specified Bar.
month	The month value of specified Bar.
day	The day value of specified Bar.
hour	The hour value of specified Bar.
minute	The minute value of specified Bar.
second	The second value of specified Bar.

5. *iBarIndex*: Required. The bar index of series to retrieve.
6. *iNumBars*: Optional. Number of bars of data to return.

Note：

getValue (barType, 0, 0) don't do any processing, returns null values.

Example

JS：

```
function OnInit() {
}
function OnTick() {
//retrieve the close from 10 bars ago
closeValue = getValue ("close", -10);
debugPrintln("closeValue: " + closeValue);
//retrieve the open of the current bar
if (! isHistoryBar ()){
openValue = getValue ("open",0);
debugPrintln("openValue "+ openValue);
}
    //retrieve the most ecent 4 high values into anarray
arrHigh =getValue ("high",0,-4);
for (var i = 0; i < 4; i++){
debugPrintln("highValue_Bar" + i + ":" + arrHigh[i]);
}
}
```

1.10 isHistoryBar

Description:

- **isHistoryBar ()**

See whether the bar is a historical data bar or not. Retrieve the BOOL value。

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
}

function OnTick() {
    if (isHistoryBar ()){
        debugPrintln ("here is history data update");
    }
    else{
        debugPrintln ("here is real-time update");
    }
}
```

1.11 getCurSymbol

Description:

- **getCurSymbol ()**

Returns the name of the symbol whose quote is being updated.
Returns an empty string when no quote is being updated.

Parameters:

None, function takes no parameters.

Example

```
function OnInit() {
```

```
EnableTrading();
}
function OnTick() {
var sym = getCurSymbol ();
debugPrintln ("sym:" + sym + "; quote:" + getQuote ("bid"));
}
```

1.12 sleep

Description:

- **sleep (*millisecond*)**

Make the script sleep for a period of the time.

Parameters:

7. *millisecond:* Required. The amount of time to sleep.

Example

JS：

```
function OnInit() {
 EnableTrading();
}
function OnTick() {
debugPrintln ("before sleep:" + Date ());
    sleep (3000);
debugPrintln ("after sleep: " + Date ());
}
```

1.13 getBidSize

Description:

- **getBidSize ()**

Retrieve current symbol’s bid size.

Parameters:

None, function takes no parameters.

Example

JS:

```
function OnInit() {
      }
function OnTick() {
      var symbol = getSymbol ();
      var bidSize = getBidSize ();
      debugPrintln (symbol + "price update, bidsize:" + bidSize);
}
```

1.14 getAskSize

Description:

- **getAskSize ()**

Retrieve current symbol's ask size.

Parameters:

None, function takes no parameters.

Example

JS:

```
function OnInit() {
    }
function OnTick() {
      var symbol = getSymbol ();
      var askSize = getAskSize ();
      debugPrintln (symbol + "price update, asksize:" + askSize);
}
```

1.15 getCurTime

Description:

- **getCurTime ()**

Returns the actual time string in "HH:MM:SS" format

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    }
function OnTick() {
      debugPrintln ("current time: " + getCurTime ());
}
```

1.16 getQuoteInfo

Description:

- **getQuoteInfo ()**

Returns the last quote object, return value is a Quote object.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
      EnableTrading();
}
function OnTick() {
var objQuote = getQuoteInfo();
debugPrintln ("symbol:" + objQuote.Symbol);
}
```

1.17 getTick

Description:

- **getTick (*iTicktype*)**

Returns the tick information for a ticktype (quotetick).
The return value is an integer value, as followed:

Return value	Comment
-2	Compared with next- to- last quote, the price is descending; but compared with last quote, the price has not changed.
-1	Compared with last quote, the price is descending
0	Two consecutive quotes without change
1	Compared with last quote, the price is ascending
2	Compared with next- to- last quote, the price is ascending; but compared with last quote, the price has not changed.

Parameters:

8. *iTicktype:* Required. A String value. The following constants are available:

Constant	Comment
0	get quote tick

Example

JS :

```
function OnInit() {
}
function OnTick() {
	var sym = getSymbol ();
	var tickVaule= getTick (0);
	debugPrintln ("sym:" + sym + ";Tick: " + tickVaule);
}
```

1.18 getBarState

Description:

- **getBarState ()**

Returns a status flag from the EFS engine indicating what bar processing is currently taking place.
The return value is an integer value, as follows:

Return value	Constant	Comment
BARSTATE_NEWBAR	0	the first tick of a new bar has arrived
BARSTATE_CURRENTBAR	1	a new tick has arrived in the current bar
BARSTATE_ALLBARS	2	script is initializing

Parameters:

None, function takes no parameters.

Example

JS:

```
function OnInit() {
}
function OnTick() {
    debugPrintln ("The bar state:" + getBarState());
    return getBarState();
}
```

1.19 getSymbol

Description:

- **getSymbol ()**

Returns the name of the symbol currently being charted.

Parameters:

None, function takes no parameters.

Example

JS:

```
function OnInit() {
}
function OnTick() {
    debugPrintln ("the symbol name:" + getSymbol ());
}
```

1.20 BarAsk

Description:

- **BarAsk ()**

Enable the given symbol to receive the quote update event from current bar.

Parameters:

In JS,0 is current bar.

Example

JS:

```
function OnInit() {

}
function OnTick() {
   var ask =   BarAsk(0);
   debugPrintln(" ask="+ask);
}
```

1.21 BarBid

Description:

BarBid ()

Enable the given symbol to receive the quote update event from current bar.

Parameters:

In JS,0 is current bar.

Example

JS:

```
function OnInit() {

}
function OnTick() {
   var bid =    BarBid(0);
   debugPrintln("bid="+bid);
}
```

1.22 getBarCount

Description:

- **getBarCount()**

Enable the funtion to calculate bars' number by the current time interval of chart on Fortex6UI.

Parameters:

None, function takes no parameters.

JS:

```
function OnInit() {

}
function OnTick() {
var bar_NO =getBarCount();
debugPrintln ("CurrentBarIndex:" + bar_NO);
}
```

1.23 getpoint

Description:

- **getPoint(*[,sSymbol]*)**

Returns the value of current symbol 1pip.

Parameters:

sSymbol **:**Optional. The symbol for which to initiate the trade

Example

JS:

```
var flag = false;
function OnInit() {

}
function OnTick() {
if(!flag){
var pip =getPoint();
debugPrintln ("The cur symbol 1 pip is " + pip);
flag=true;
}
}
```

2. Chart data processing

2.1 Price Functions

2.1.1 high

Description:

- **high (***iBarIndex* [,*iNumBars*]**)**

Returns the high price at the specified bar index.

Parameters:

9. *iBarIndex:* Required. The Bar index of series to retrieve.
10. *iNumBars*: Optional. Number of bars of data to return.

Note：

high (0, 0) not done any processing, returns null.

Example

JS:

```
function OnInit() {
}
function OnTick() {
    if (! isHistoryBar ()){
        //retrieve the value for the current price bar
        var HighValue1 = high (0);
        debugPrintln (" current high value: " + HighValue1);
    }
    //retrieve the value for the previous price bar
    var HighValue2 = high (-1);
    debugPrintln ("previous high value:" + HighValue2);
    //retrieve the last 4 prices into an array
    var arrHigh = high (0, -4);
    for (var i = 0; i < 4; i++){
        debugPrintln ("Last HighValue_Bar" + i + ":" + arrHigh[i]);
    }
}
```

2.1.2 low

Description:

- **low (***iBarIndex* [,*iNumBars*]**)**

Returns the low price at the specified bar index.

Parameters:

11. *iBarIndex:* Required. The Bar index of series to retrieve.
12. *iNumBars*: Optional. Number of bars of data to return.

Note：

low (0, 0) not done any processing, returns null.

Example

JS:

```
function OnInit() {
}
function OnTick() {
    if (! isHistoryBar ()){
        //retrieve the value for the current price bar
        var LowValue1 = low (0);
        debugPrintln ("current Low value: " + LowValue1);
    }
    //retrieve the value for the previous price bar
    var LowValue2 = low (-1);
    debugPrintln ("previous Low value:" + LowValue2);
    //retrieve the last 4 prices into an array
    var arrLow = low (0, -4);
    for (var i = 0; i < 4; i++){
        debugPrintln ("Last LowValue_Bar" + i + ":" + arrLow[i]);
    }
}
```

2.1.3 close

Description:

- **close (***iBarIndex* [,*iNumBars*]**)**

Returns the close price at the specified bar index.

Parameters:

13. *iBarIndex:* Required. The Bar index of series to retrieve.
14. *iNumBars*: Optional. Number of bars of data to return.

Note：

close (0, 0) not done any processing, returns null.

Example

JS:

```
function OnInit() {
}
function OnTick() {
    if (! isHistoryBar ()){
        //retrieve the value for the current price bar
        var CloseValue1 = close (0);
        debugPrintln ("current Low value: " + CloseValue1);
    }
    //retrieve the value for the previous price bar
    var CloseValue2 = close (-1);
    debugPrintln ("previous Low value:" + CloseValue2);
    //retrieve the last 4 prices  into an array
    var arrClose = close (0, -4);
    for (var i = 0; i < 4; i++){
        debugPrintln ("Last CloseValue_Bar" + i + ":" + arrClose[i]);
    }
}
```

2.1.4 open

Description:

- **open (***iBarIndex* [,*iNumBars*])

Returns the open price at the specified bar index.

Parameters:

15. *iBarIndex:* Required. The Bar index of series to retrieve.
16. *iNumBars*: Optional. Number of bars of data to return.

Note：

open (0, 0) not done any processing, returns null.

Example

```
function OnInit() {
}
function OnTick() {
    if (! isHistoryBar ()){
        //retrieve the value for the current price bar
        var OpenValue1 = open (0);
        debugPrintln (" current Open value: " + OpenValue1);
    }
    //retrieve the value for the previous price bar
    var OpenValue2 = open (-1);
    debugPrintln ("previous Open value:" + OpenValue2);
    //retrieve the last 4 prices  into an array
    var arrOpen = open (0, -4);
    for (var i = 0; i < 4; i++){
        debugPrintln ("Last OpenValue_Bar" + i + ":" + arrOpen[i]);
    }
}
```

2.1.5 highest

Description:

- **highest (***iNumbars, iSeries***)**

This function will return the highest value found in a series within numBars number of bars.

Parameters:

17. *iNumbars:* Required. The number of bars back to search.
18. *iSeries:* Required. The series in which to search for the value.

Example

```
function OnInit() {
}
function OnTick() {
    if (! isHistoryBar ()){
        //find the highest-high over the last 50 bars
        var HighestValue = highest (50, high ());
```

```
            debugPrintln ("HighestValue" + HighestValue);
        }
    }
```

2.1.6 hhv

Description:

- **hhv (*iNumbars, iSeries*)**

This function will return the highest value found in a series within numBars number of bars.

Parameters:

19. *iNumbars:* Required. The number of bars back to search.
20. *iSeries:* Required. The series in which to search for the value.

Example

JS：

```
function OnInit() {
}
function OnTick() {
    if (! isHistoryBar ()){
        //find the highest-high over the last 50 bars
        var HighestValue = hhv (50, high ());
        debugPrintln ("HighestValue" + HighestValue);
    }
}
```

2.1.7 llv

Description:

- **llv (*iNumbars, iSeries*)**

This function will return the lowest value found in a series within numBars number of bars.

Parameters:

21. *iNumbars:* Required. The number of bars back to search.

22. *iSeries:* Required. The series in which to search for the value.

Example

JS：

```
function OnInit() {
}
function OnTick() {
    if (! isHistoryBar ()){
        //find the lowest-low over the last 50 bars
        var LowestValue = llv (50, high ());
        debugPrintln ("LowestValue:" + LowestValue);
    }
}
```

2.1.8 lowest

Description:

- **lowest (***iNumbars, iSeries***)**

This function will return the lowest value found in a series within numBars number of bars.

Parameters:

23. *iNumbars:* Required. The number of bars back to search.
24. *iSeries:* Required. The series in which to search for the value.

Example

JS:

```
function OnInit() {
}
function OnTick() {
    if (! isHistoryBar ()){
        //find the lowest-low over the last 50 bars
        var LowestValue = lowest (50, low ());
        debugPrintln ("LowestValue" + LowestValue);
    }
}
```

2.2 Time Functions

2.2.1 year

Description:

- **year (***iBarIndex* [,*iNumBars*]**)**

Returns the year at the specified bar index.

Parameters:

25. *iBarIndex:* Required. The Bar index of series to retrieve.
26. *iNumBars*: Optional. Number of bars of data to return.

Note：

year (0, 0) not done any processing, returns null.

Example

JS:

```
function OnInit() {
}
function OnTick() {
    if (! isHistoryBar ()){
        //retrieve the value for the current price bar
        var strYear1 = year (0);
        debugPrintln ("current year: " + strYear1);
    }
    //retrieve the value for the previous price bar
    var strYear2 = year (-1);
    debugPrintln ("previous year:" + strYear2);
    //retrieve the last 4 values into an array
    var arrYear = year (0, -4);
    for (var i = 0; i < 4; i++){
        debugPrintln ("Last Year_Bar" + i + ":" + arrYear[i]);
    }
}
```

2.2.2 month

Description:

- **month (***iBarIndex* [,*iNumBars*]**)**

Returns the month at the specified bar index.

Parameters:

27. *iBarIndex:* Required. The Bar index of series to retrieve.
28. *iNumBars*: Optional. Number of bars of data to return.

Note：

month (0, 0) not done any processing, returns null.

Example

JS:

```
function OnInit() {
}
function OnTick() {
    if (! isHistoryBar ()){
        //retrieve the value for the current price bar
        var strMonth1 = month (0);
        debugPrintln ("current month: " + strMonth1);
    }
    //retrieve the value for the previous price bar
    var strMonth2 = month (-1);
    debugPrintln ("previous month:" + strMonth2);
    //retrieve the last 4 values into an array
    var arrMonth = month (0, -4);
    for (var i = 0; i < 4; i++){
        debugPrintln ("Last Month_Bar" + i + ":" + arrMonth[i]);
    }
}
```

2.2.3 day

Description:

- **day (***iBarIndex* [,*iNumBars*]**)**

Returns the day at the specified bar index.

Parameters:

29. *iBarIndex:* Required. The Bar index of series to retrieve.
30. *iNumBars*: Optional. Number of bars of data to return.

Note：

day (0, 0) not done any processing, returns null.

Example

JS :

```
function OnInit() {
}
function OnTick() {
    if (! isHistoryBar ()){
        //retrieve the value for the current price bar
        var strDay1 = day(0);
        debugPrintln ("current day: " + strDay1);
    }
    //retrieve the value for the previous price bar
    var strDay2 = day (-1);
    debugPrintln ("previous day:" + strDay2);
    //retrieve the last 4 values into an array
    var arrDay = day (0, -4);
    for (var i = 0; i < 4; i++){
        debugPrintln ("Last Day_Bar" + i + ":" + arrDay [i]);
    }
}
```

2.2.4 hour

Description:

- **hour (***iBarIndex* [,*iNumBars*]**)**

Returns the hour at the specified bar index.

Parameters:

31. *iBarIndex:* Required. The Bar index of series to retrieve.
32. *iNumBars*: Optional. Number of bars of data to return.

Note：

hour (0, 0) not done any processing, returns null.

Example

JS:

```
function OnInit() {
}
function OnTick() {
    if (! isHistoryBar ()){
        //retrieve the value for the current price bar
        var strHour1 = hour (0);
        debugPrintln ("current hour: " + strHour1);
    }
    //retrieve the value for the previous price bar
    var strHour2 = hour (-1);
    debugPrintln ("previous hour:" + strHour2);
    //retrieve the last 4 values into an array
    var arrHour = hour (0, -4);
    for (var i = 0; i < 4; i++){
        debugPrintln ("Last Hour_Bar" + i + ":" + arrHour[i]);
    }
}
```

2.2.5 minute

Description:

- **minute (***iBarIndex* [,*iNumBars*]**)**

Returns the minute at *the specified bar index.*

Parameters:

33. *iBarIndex:* Required. The Bar index of series to retrieve.
34. *iNumBars*: Optional. Number of bars of data to return.

Note：

minute (0, 0) not done any processing, returns null.

Example

JS:

```
function OnInit() {
}
function OnTick() {
    if (! isHistoryBar ()){
        //retrieve the value for the current price bar
        var strMinute1 = minute (0);
        debugPrintln ("current minute: " + strMinute1);
    }
    //retrieve the value for the previous price bar
    var strMinute2 = minute (-1);
    debugPrintln ("previous minute:" + strMinute2);
    //retrieve the last 4 values into an array
    var arrMinute = minute (0, -4);
    for (var i = 0; i < 4; i++){
        debugPrintln ("Last Minute_Bar" + i + ":" + arrMinute[i]);
    }
}
```

2.2.6 second

Description:

- **second (***iBarIndex* [,*iNumBars*]**)**

Returns the second at the specified bar index.

Parameters:

35. *iBarIndex:* Required. The Bar index of series to retrieve.
36. *iNumBars*: Optional. Number of bars of data to return.

Note：

second (0, 0) not done any processing, returns null.

Example

JS:

```
function OnInit() {
}
function OnTick() {
if (! isHistoryBar ()){
        //retrieve the value for the current price bar
       var strSecond1 = second (0);
```

```
            debugPrintln ("current second: " + strSecond1);
    }
    //retrieve the value for the previous price?bar
       var strSecond2 = second (-1);
       debugPrintln ("previous second:" + strSecond2);
    //retrieve the last 4 values into an array
      var arrSecond = second (0, -4);
    for (var i = 0; i < 4; i++){
    debugPrintln ("Last Second_Bar" + i + ":" + arrSecond[i]);

    }
    }
```

2.2.7 getYear

Description:

- **getYear (***iBarIndex* [,*iNumBars*]**)**

Returns the year at the specified bar index.

Parameters:

37. *iBarIndex:* Required. The Bar index of series to retrieve.
38. *iNumBars*: Optional. Number of bars of data to return.

Note：

getYear (0, 0) not done any processing, returns null.

Example

JS:

```
    function OnInit() {
    }
    function OnTick() {
        if (! isHistoryBar ()){
                //retrieve the value for the current price bar
                var strYear1 = getYear (0);
                debugPrintln ("current year: " + strYear1);
         }
         //retrieve the value for the previous price bar
         var strYear2 = getYear (-1);
         debugPrintln ("previous year:" + strYear2);
```

```
        //retrieve the last 4 prices into an array
        var arrYear = getYear (0, -4);
        for (var i = 0; i < 4; i++){
            debugPrintln ("Last Year_Bar" + i + ":" + arrYear[i]);
        }
    }
```

2.2.8 getMonth

Description:

- **getMonth (***iBarIndex* [,*iNumBars*]**)**

Returns the month at the specified bar index.

Parameters:

39. *iBarIndex:* Required. The Bar index of series to retrieve.
40. *iNumBars*: Optional. Number of bars of data to return.

Note：

getMonth (0, 0) not done any processing, returns null.

Example

JS:

```
    function OnInit() {
    }
    function OnTick() {
        if (! isHistoryBar ()){
            //retrieve the value for the current price bar
            var strMonth1 = getMonth (0);
            debugPrintln ("current month: " + strMonth1);
        }
        //retrieve the value for the previous price bar
        var strMonth2 = getMonth (-1);
        debugPrintln ("previous month:" + strMonth2);
        //retrieve the last 4 values into an array
        var arrMonth = getMonth (0, -4);
        for (var i = 0; i < 4; i++){
            debugPrintln ("Last Month_Bar" + i + ":" + arrMonth[i]);
        }
    }
```

2.2.9 getDay

Description:

- **getDay (***iBarIndex* [,*iNumBars*]**)**

Returns the day at the specified bar index.

Parameters:

41. *iBarIndex:* Required. The Bar index of series to retrieve.
42. *iNumBars*: Optional. Number of bars of data to return.

Note：

getDay (0, 0) not done any processing, returns null.

Example

JS：

```
function OnInit() {
}
function OnTick() {
    if (! isHistoryBar ()){
        //retrieve the value for the current price bar
        var strDay1 = getDay (0);
        debugPrintln ("current day: " + strDay1);
    }
    //retrieve the value for the previous price bar
    var strDay2 = getDay (-1);
    debugPrintln ("previous day:" + strDay2);
    //retrieve the last 4 prices into an array
    var arrDay = getDay (0, -4);
    for (var i = 0; i < 4; i++){
        debugPrintln ("Last Day_Bar" + i + ":" + arrDay [i]);
    }
}
```

2.2.10 getMinute

Description:

- **getMinute (***iBarIndex* [,*iNumBars*]**)**

Returns the minute at the specified bar index.

Parameters:

43. *iBarIndex:* Required. The Bar index of series to retrieve.
44. *iNumBars*: Optional. Number of bars of data to return.

Note：

getMinute (0, 0) not done any processing, returns null.

Example

JS：

```
function OnInit() {
}
function OnTick() {
    if (! isHistoryBar ()){
        //retrieve the value for the current price bar
        var strMinute1 = getMinute (0);
        debugPrintln ("current minute: " + strMinute1);
    }
    //retrieve the value for the previous price bar
    var strMinute2 = getMinute (-1);
    debugPrintln ("previous minute:" + strMinute2);
    //retrieve the last 4 values into an array
    var arrMinute = getMinute (0, -4);
    for (var i = 0; i < 4; i++){
        debugPrintln ("Last Minute_Bar" + i + ":" + arrMinute[i]);
    }
}
```

2.2.11 getSecond

Description:

- **getSecond (***iBarIndex* [,*iNumBars*]**)**

Returns the second at the specified bar index.

Parameters:

45. *iBarIndex:* Required. The Bar index of series to retrieve.
46. *iNumBars*: Optional. Number of bars of data to return.

Note：

getSecond (0, 0) not done any processing, returns null.

Example

JS:

```
function OnInit() {
}
function OnTick() {
    if (! isHistoryBar ()){
        //retrieve the value for the current price bar
        var strSecond1 = getSecond (0);
        debugPrintln ("current second: " + strSecond1);
    }
    //retrieve the value for the previous price bar
    var strSecond2 = getSecond (-1);
    debugPrintln ("previous second:" + strSecond2);
    //retrieve the last 4 values into an array
    var arrSecond = getSecond (0, -4);
    for (var i = 0; i < 4; i++){
        debugPrintln ("Last Second_Bar" + i + ":" + arrSecond[i]);
    }
}
```

3. Built-In Study Functions

3.1 Average True Range

Description:

- **atr (*length*)**

The **Average True Range** (ATR) is a measure of volatility. It was introduced by Welles Wilder in his book New Concepts in Technical Trading Systems and has since been used as a component of many indicators and trading systems.
Wilder has found that the high ATR values often occur at market bottoms following a "panic" sell-off. Low ATR values are often found during extended sideways periods, such as those found at tops and after consolidation periods.

Parameters:

47. *length:* Required, period to use for the calculation.

Example

JS:

```
var xATR=null;
var max =0;
var Length;
var atrLine;

function OnInit() {
   setPriceStudy(false);
   Length = new FunctionParameter("Length",FunctionParameter.INT,14);
    atrLine=new IndicatorLine("ATR",Colors.Yellow);
   xATR=atr(Length.value());

   SetTitle("ATR("+Length.value()+")");
}


function OnTick(){
      var result= xATR.valueOf();
      atrLine.Refresh(result);


}
```

Description:

- **upperBB (***length*, *stdDev* [, *source*] [, *InitIndex*])
- **middleBB (***length*, *stdDev* [, *source*] [, *InitIndex*])
- **lowerBB (***length*, *stdDev* [, *source*] [, *InitIndex*])

3.2 Bollinger Bands

Bollinger Bands (created by John Bollinger) are similar to moving average envelopes. The difference between Bollinger Bands and envelopes is that envelopes are plotted at a fixed percentage above and below a moving average, whereas Bollinger Bands are plotted at standard deviation levels above and below a moving average. Since standard deviation is a measure of volatility, the bands are self-adjusting, widening during volatile markets and contracting during calmer periods.

Parameters:

48. *length:* Required. The period to use for the calculation.
49. *stdDev:* Required. The number of standard deviations.
50. *source*: Optional. Input series for the study.
    Default: close
51. *InitIndex:* Optional. The bar index of value to retrieve.
    Default: 0, current value

Example

JS:

```
    var bollinger =null;
    var PriceSource;
    var Length;
    var StdDev;
    var upperBBLine;
    var middleBBLine;
    var lowerBBLine

    function OnInit() {

        upperBBLine = new IndicatorLine("Upper",Colors.Yellow);
        middleBBLine = new IndicatorLine("Middle",Colors.Red);
        lowerBBLine = new IndicatorLine("Lower",Colors.Green);

    PriceSource=new
    FunctionParameter("PriceSource",FunctionParameter.OPTION,MarketSeries.O
PEN,"markettype");
         Length= new FunctionParameter("Lenght",FunctionParameter.INT,20);
         StdDev= new FunctionParameter("StdDev",FunctionParameter.INT,2);

        SetTitle("Bollinger("+Length.value()+","+StdDev.value()+")");
    bollinger=new BollingerStudy (Length.value(), StdDev.value(),
PriceSource.value());

    }

    function OnTick() {
        var upper = bollinger.getValue(BollingerStudy.UPPER).valueOf();
        var middle =bollinger.getValue(BollingerStudy.BASIS).valueOf();
        var lower=bollinger.getValue(BollingerStudy.LOWER).valueOf();


        upperBBLine.Refresh(upper);
        middleBBLine.Refresh(middle);
```

```
        lowerBBLine.Refresh(lower);
    }
```

3.3 Commodity Channel Index

Description:

- **cci (***length* [, *source*] [, *InitIndex*]**)**

The **Commodity Channel Index** (CCI) is a price momentum indicator that measures the price excursions from the mean price as a statistical variation. It is used to detect the beginnings and endings of trends.

Parameters:

52. *length:* Required. The period to use for the calculation.
53. *source* Optional. Input series for the study.
Default: close
54. *InitIndex:* Optional. The bar index of value to retrieve.
Default: 0, current value。

Example

JS:

```
    var CCI=null;
    var Period=10;
    var Line;
    var PriceSource;
    function OnInit() {
      setPriceStudy(false);
      Line=new IndicatorLine("CCI",Colors.Blue);
    PriceSource=new                                    FunctionParameter("Price
Source",FunctionParameter.OPTION,MarketSeries.CLOSE,"markettype");
      CCI=cci(Period, PriceSource.value());
      SetTitle("CCI(10)");
    }
    function OnTick(){
        var result= CCI.getValue (0);
          Line.Refresh(result);
    }
```

3.4 Moving Average

Description:

- **ema (***length* [, *source*] [, *InitIndex*]**)**
- **sma (***length* [, *source*] [, *InitIndex*]**)**
- **wma (***length* [, *source*] [, *InitIndex*]**)**
- **vwma (***length* [, *source*] [, *InitIndex*]**)**

A **Moving Average** is an indicator that shows the average value of a security's price over a period of time. When calculating a moving average, a mathematical analysis of the security's average value over a predetermined time period is made. As the security's price changes, its average price moves up or down.

Parameters:

55. *length:* Required. The period to use for the calculation.
56. *source*: Optional. Input series for the study.
Default: close
57. *InitIndex:* Optional. The bar index of value to retrieve.
Default: 0, current value.

Example

JS:

```
var emaStudy;
var  emaLine;
var Length;
var PriceSource;

function OnInit() {
    emaLine=new IndicatorLine("EMA",Colors.Yellow);
    Length= new FunctionParameter("Length",FunctionParameter.INT,10);
PriceSource= new FunctionParameter("Price Source",FunctionParameter.OPTION,MarketSeries.CLOSE,"markettype");

    emaStudy = ema(Length.value(), PriceSource.value());
    SetTitle("EMA("+Length.value()+")");
}
function OnTick(index) {
```

```
        var    val = emaStudy.getValue (0);
        emaLine.Refresh(val);

    }
```

3.5 MACD

Description:

- **macd (***fastLength*, *slowLength*, *smoothing* [, *source*] [, *barIndex*]**)**
- **macdSignal (***fastLength*, *slowLength*, *smoothing* [, *source*] [, *barIndex*]**)**
- **macdHist (***fastLength*, *slowLength*, *smoothing* [, *source*] [, *barIndex*]**)**

MACD is short for Moving Average Convergence Divergence. The MACD looks at the difference between a short-term moving average and a long-term moving average. A third moving average is taken of the difference and used as a signal line.

Parameters:

58. *fastLength:* Required. the fast MACD period.
59. *slowLength:* Required. the slow MACD period.
60. *smoothing:* the MACD smoothing period.
61. *source*: Optional. Input series for the study.
Default: close
62. *InitIndex:* Optional. The bar index of value to retrieve.
Default: 0, current value

Example

JS:

```
    var macd;
    var macdLine;
    var macdSignalLine;
    var macdHistLine;
    var zeroLine;
    var Fast;
    var Slow;
    var Smoothing;

    function OnInit() {
```

```
                setPriceStudy(false);
                macdLine=new IndicatorLine("Macd",Colors.Blue);
                macdSignalLine=new IndicatorLine("Signal",Colors.Red);
                macdHistLine=new IndicatorLine("Hist",Colors.Yellow,LineType.Bar);
                zeroLine=new IndicatorLine("Zero",Colors.White);

                 Fast= new FunctionParameter("Fast",FunctionParameter.INT,12);
                 Slow= new FunctionParameter("Slow",FunctionParameter.INT,26);
                                                Smoothing=                    new
FunctionParameter("Smoothing",FunctionParameter.INT,9);

SetTitle("MACD("+Fast.value()+","+Slow.value()+","+Smoothing.value()+")");
                macd =new MACDStudy(Fast.value(), Slow.value(), Smoothing.value());
            }

            function OnTick(index) {
               var macdValue= macd.getValue(MACDStudy.MACD).valueOf();
               var signalValue =macd.getValue(MACDStudy.SIGNAL).valueOf();
               var histValue =macd.getValue(MACDStudy.HIST).valueOf();


               macdLine.Refresh(macdValue);
               macdSignalLine.Refresh(signalValue);
               macdHistLine.Refresh(histValue);
               zeroLine.Refresh(0);
            }
```

3.6 Rate of Change

Description:

- **roc (***length* [, *source*] [, *InitIndex*]**)**

The **Rate-of-Change** (ROC) indicator displays the difference between the current price and the price x time periods ago. The difference can be displayed either in points or as a percentage. The momentum indicator displays the same information but expresses it as a ratio.

Parameters:

63. *length: Required. The period to use for the calculation.*
64. *source: Optional. Input series for the study.*

65. *Default: close*

66. *InitIndex: Optional. The bar index of value to retrieve.*

67. *Default: 0, current value.*

Example

JS:

```
var ROC=null;
var Period;
var PriceSource;
var Line;

function OnInit() {
   setPriceStudy(false);
   Period = new FunctionParameter("Period",FunctionParameter.INT,9);
   PriceSource= new FunctionParameter("Price
Source",FunctionParameter.OPTION,MarketSeries.OPEN,"markettype");
   Line=new IndicatorLine("Roc",Colors.Blue);
   ROC=roc(Period.value(),PriceSource.value());

   SetTitle("ROC("+Period.value()+")");
}


function OnTick(){
      var result= ROC.getValue (0);
        Line.Refresh(result);

}
```

4. Trading Functions

4.1 buyMarket

Description:

- **buyMarket(***iQuantity [,sSymbol] [,dSL][,dTP]*[,*sAccountName*] [,*sExpiry*]**)**

Submit a buy market order to broker.

Parameters:

iQuantity: Required, a positive integer value. The number of shares/contracts.
dSL: Optional,StopLoss price.
dTP: Optional,TakeProfit price.
sAccountName: Optional. The account name for which to initiate the trade.
sSymbol: Optional. The symbol for which to initiate the trade
sExpiry: Optional. Expiration for the order. The following constants are available:

Constant	Comment
FOK	TIF Setting is “FOK”
GTC	TIF Setting is “GTC”
IOC	TIF Setting is “IOC”

Example ：

JS1:

```
var isSend = false;
function OnInit() {
  EnableTrading();

}
function OnTick() {
      if (! isSend) {
            if (! isHistoryBar()) {
                  buyMarket (10000);
                  debugPrintln ("buy");
                  isSend = true;
            }
      }
}
```

JS2：

```
var isSend = false;
function OnInit() {
        EnableTrading();
}
function OnTick() {
      if (! isSend){
            if (! isHistoryBar()){
                  buyMarket (21000,"EUR/USD",1.21,1.31,"FX01","GTC");
                   debugPrintln ("buy, enter all parameter");
                   isSend = true;
            }
      }
}
```

4.2 buyLimit

Description:

- **buyLimit (***iQuantity*, *dLimitPrice [,sSymbol] [,dSL][,dTP][,sAccountName] [,sExpiry]***)**

Submit a buy limit order to broker.

Parameters:

68. *iQuantity*: Required, a positive integer value. The number of shares/contracts.
69. *dSL: Optional,StopLoss price.*
70. *dTP: Optional,TakeProfit price.*
71. *dLimitPrice*: Required. The limit price.
72. *sAccountName:* Optional. The account name for which to initiate the UserScript.
73. *sSymbol:* Optional. The symbol for which to initiate the trade
74. *sExpiry:* Optional. Expiration for the order. The following constants are available:

Constant	Comment
FOK	TIF Setting is “FOK”
GTC	TIF Setting is “GTC”
IOC	TIF Setting is “IOC”

Example ：

JS1:

```
var isSend = false;
function OnInit() {
    EnableTrading();

}
function OnTick() {
    if (! isSend) {
        var limit_price = getQuote ("ask") - 0.02;
        if (! isHistoryBar ()){
            buyLimit (12000,limit_price);
            debugPrintln ("buy limit");
            isSend = true;
        }
    }
}
```

JS2：

```
var isSend = false;
function OnInit() {
    EnableTrading();

}
function OnTick() {
    if (! isSend) {
        var limit_price = getQuote ("ask","EUR/JPY") - 0.02;
        if (! isHistoryBar ()){
        buyLimit (19000,limit_price,"EUR/JPY",142.2,143.3,"FX01","GTC");
            debugPrintln ("buy limit");
            isSend = true;
        }
    }
}
```

4.3 buyStop

Description:

- **buyStop (***iQuantity*, *dStopPrice* [,*sSymbol*] *[,dSL][,dTP]*[,*sAccountName*] [,*sExpiry*]**)**

Submit a buy stop order to broker.

Parameters:

75. *iQuantity*: Required, a positive integer value. The number of shares/contracts.
76. *dStopPrice:* Required. The stop price.
77. *dSL: Optional,StopLoss price.*
78. *dTP: Optional,TakeProfit price.*
79. *sAccountName:* Optional. The account name for which to initiate the trade.
80. *sSymbol:* Optional. The symbol for which to initiate the trade
81. *sExpiry:* Optional. Expiration for the order. The following constants are available:

Constant	Comment
FOK	TIF Setting is “FOK”
GTC	TIF Setting is “GTC”
IOC	TIF Setting is “IOC”

Example：

JS1:

```
var isSend = false;
function OnInit() {
    EnableTrading();

}
function OnTick() {
    if (! isSend) {
        var stop_price = getQuote ("ask") + 0.02;
        if (! isHistoryBar ()){
            buyStop (10000, stop_price);
            debugPrintln ("buy stop");
            isSend = true;
        }
    }
}
```

JS2：

```
var isSend = false;
function OnInit() {
    EnableTrading();

}
function OnTick() {
    if (! isSend){
        var stop_price = getQuote ("ask") + 0.02;
        if (! isHistoryBar ()){
            buyStop (15000, stop_price,"GBP/CAD",0,0,"FX01", "GTC");
            debugPrintln ("buy stop,enter all parameter");
            isSend = true;
        }
    }
}
```

4.4 buyThreshold

Description:

- **buyThreshold (***iQuantity*, *dLowerlimit*, *dUpperLimit [,sSymbol]* [,*sAccountName*] [,*sExpiry*]**)**

Submit a buy threshold order to broker.

Parameters:

82. *iQuantity*: Required, a positive integer value. The number of shares/contracts.
83. *dLowerlimit:* Required. The lower price.
84. *dUpperLimit:* Required. The upper price.
85. *sAccountName:* Optional. The account name for which to initiate the trade.
86. *sSymbol:* Optional. The symbol for which to initiate the trade
87. *sExpiry:* Optional. Expiration for the order. The following constants are available:

Constant	Comment
FOK	TIF Setting is “FOK”
GTC	TIF Setting is “GTC”
IOC	TIF Setting is “IOC”

Example

JS:

```
var isSend = false;
function OnInit() {
    EnableTrading();

}
function OnTick() {
    if (! isSend) {
        var upper_price = getQuote ("ask") + 0.02;
        var lower_price = getQuote ("ask")   - 0.05;
        if (! isHistoryBar ()){
            buyThreshold (10000, lower_price, upper_price);
            debugPrintln ("buy Threshold");
            isSend = true;
        }
    }
}
```

JS2:

```
var isSend = false;
function OnInit() {
    EnableTrading();
}
function OnTick() {
    if (! isSend) {
        var lower_price = getQuote ("ask","AUD/CAD") - 0.05;
        var upper_price = getQuote ("ask","AUD/CAD") + 0.05;
        if (! isHistoryBar()){
buyThreshold(14000, lower_price, upper_price,"AUD/CAD","FX01","GTC");
```

```
                debugPrintln ("buy Threshold");
                isSend = true;
                }
        }
}
```

4.5 buy

Description:

- **buy (***[sAccountName] , [sSymbol]***)**

Submit a market order to broker using the pre-set default number of shares/contracts.

Parameters:

88. *sAccountName:* Optional. The account name for which to initiate the trade.
89. *sSymbol:* Optional. The symbol for which to initiate the trade

Example ：

JS1：

```
var isSend = false;
function OnInit() {
        EnableTrading();

        var intQuan = getDefaultQuantity();
        if(intQuan<1000){
            setDefaultQuantity(10000);
        }
}
function OnTick() {
        if (! isSend) {
            if (! isHistoryBar ()){
                buy ();
                debugPrintln ("buy");
                isSend = true;
            }
        }
}
```

JS2：

```
var isSend = false;
```

```
function OnInit() {
    EnableTrading();

    var intQuan = getDefaultQuantity();
    if(intQuan<1000){
        setDefaultQuantity(10000);
    }
}
function OnTick() {
    if (! isSend){
        if(!isHistoryBar()){
            buy ("FX01","AUD/CAD");
            debugPrintln("buy,enter all parameter");
            isSend=true;
        }
    }
}
```

4.6 sellMarket

Description:

- **sellMarket (***iQuantity, [,sSymbol] [,dSL][,dTP][,sAccountName] [,sExpiry]***)**

Submit a sell market order to broker.

Parameters:

90. *iQuantity*: Required, a positive integer value. The number of shares/contracts.
91. *dSL: Optional,StopLoss price.*
92. *dTP: Optional,TakeProfit price.*
93. *sAccountName:* Optional. The account name for which to initiate the trade.
94. *sSymbol:* Optional. The symbol for which to initiate the trade.
95. *sExpiry:* Optional. Expiration for the order. The following constants are available:

Constant	Comment
FOK	TIF Setting is “FOK”
GTC	TIF Setting is “GTC”
IOC	TIF Setting is “IOC”

Example

JS1:

```
var isSend = false;
```

```
function OnInit(){
    EnableTrading();

}
function OnTick() {
    if(!isSend) {
        if (!isHistoryBar()) {
            sellMarket (1000);
            debugPrintln("sell");
            isSend = true;
        }
    }
}
```

JS2：

```
var isSend = false;
function OnInit(){
EnableTrading();}
function OnTick() {
    if(!isSend) {
        if (!isHistoryBar()) {
            sellMarket (14000,"EUR/JPY",143.9,138.5,"FX01","GTC");
            debugPrintln("sell");
            isSend = true;
        }
    }
}
```

4.7 sellLimit

Description:

- **sellLimit(***iQuantity,dLimitPrice[,sSymbol] [,dSL][,dTP][,sAccountName] [,sExpiry]***)**

Submit a sell limit order to broker.

Parameters:

96. *iQuantity*: Required, a positive integer value. The number of shares/contracts.
97. *dLimitPrice*: Required. The limit price.
98. *dSL: Optional,StopLoss price.*
99. *dTP: Optional,TakeProfit price.*
100. *sAccountName:* Optional. The account name for which to initiate the trade.

101. *sSymbol:* Optional. The symbol for which to initiate the trade

102. *sExpiry:* Optional. Expiration for the order. The following constants are available:

Constant	Comment
FOK	TIF Setting is “FOK”
GTC	TIF Setting is “GTC”
IOC	TIF Setting is “IOC”

Example

JS1:

```
var isSend = false;
function OnInit(){
    EnableTrading();

}
function OnTick() {
    if(! isSend) {
        var limit_price = getQuote("bid") + 0.02;
        if(!isHistoryBar()){
            sellLimit (10000, limit_price);
            debugPrintln ("sell limit");
            isSend = true;
        }
    }
}
```

JS2：

```
var isSend = false;
function OnInit(){
    EnableTrading();
}
function OnTick() {
    if(! isSend) {
        var limit_price = getQuote("bid","EUR/JPY") + 0.02;
        if(!isHistoryBar()){
            sellLimit(12000,limit_price,"EUR/JPY",143.5,138.5,"FX01","GTC");
            debugPrintln ("sell limit");
            isSend = true;
        }
    }
}
```

4.8 sellStop

Description:

- **sellStop(***iQuantity,dStopPrice[,sSymbol][,dSL][,dTP][,sAccountName][,sExpiry]***)**

Submit a sell stop order to broker.

Parameters:

103.*iQuantity*: Required, a positive integer value. The number of shares/contracts.
104.*dStopPrice:* Required. The stop price.
105.*dSL: Optional,StopLoss price.*
106.*dTP: Optional,TakeProfit price.*
107.*sAccountName:* Optional. The account name for which to initiate the trade.
108.*sSymbol:* Optional. The symbol for which to initiate the trade
109.*sExpiry:* Optional. Expiration for the order. The following constants are available:

Constant	Comment
FOK	TIF Setting is “FOK”
GTC	TIF Setting is “GTC”
IOC	TIF Setting is “IOC”

Example

JS1:

```
var isSend = false;
function OnInit() {
    EnableTrading();

}
function OnTick() {
    if(!isSend) {
        var stop_price = getQuote ("bid") - 0.02;
        if (!isHistoryBar()){
            sellStop (16000, stop_price);
            debugPrintln ("sell stop");
            isSend = true;
        }
    }
}
```

JS2：

```
var isSend = false;
function OnInit() {
    EnableTrading();
}
function OnTick() {
    if(!isSend) {
        var stop_price = getQuote("bid","EUR/JPY") - 0.02;
        if(!isHistoryBar()){
            sellStop(16000,stop_price,"EUR/JPY",145.5,133.5,"FX01","GTC");
            debugPrintln ("sell stop");
            isSend = true;
        }
    }
}
```

4.9 sellThreshold

Description:

- **sellThreshold** *(iQuantity, dLowerlimit, dUpperLimit [,sAccountName] [,sSymbol] [,sExpiry])*

Submit a sell threshold order to broker.

Parameters:

110. *iQuantity*: Required, a positive integer value. The number of shares/contracts.
111. *dLowerlimit:* Required. The lower price.
112. *dUpperLimit:* Required. The upper price.
113. *sAccountName:* Optional. The account name for which to initiate the trade.
114. *sSymbol:* Optional. The symbol for which to initiate the trade
115. *sExpiry:* Optional. Expiration for the order. The following constants are available:

Constant	Comment
FOK	TIF Setting is “FOK”
GTC	TIF Setting is “GTC”
IOC	TIF Setting is “IOC”

Example

JS1:

```
var isSend = false;
function OnInit() {
    EnableTrading();
}
```

```
function OnTick() {
    if (!isSend){
        var upper_price = getQuote("bid") + 0.02;
        var lower_price = getQuote("bid") - 0.05;
        if (!isHistoryBar()){
    sellThreshold (12200, lower_price, upper_price);
            debugPrintln("sell Threshold");
            isSend = true;
        }
    }
}
```

JS2：

```
var isSend = false;
function OnInit() {
        EnableTrading();
}
function OnTick() {
    if (!isSend){
        var upper_price = getQuote("bid","EUR/JPY") + 0.02;
        var lower_price = getQuote("bid","EUR/JPY") - 0.05;
        if (!isHistoryBar()){
    sellThreshold (12200, lower_price, upper_price,"EUR/JPY","FX01","GTC");
            debugPrintln("sell Threshold");
            isSend = true;
        }
    }
}
```

4.10 sell

Description:

- **sell ([***sAccountName***] [,***sSymbol***])**

Submit a market order to broker using the pre-set default number of shares/contracts.

Parameters:

116. *sAccountName:* Optional. The account name for which to initiate the trade.
117. *sSymbol:* Optional. The symbol for which to initiate the trade.

Example

JS1：

```
var isSend = false;
function OnInit() {
EnableTrading();
    var intQuan = getDefaultQuantity();
    if(intQuan<1000){
        setDefaultQuantity(10000);
    }
}
function OnTick(){
    if (!isSend) {
        if (!isHistoryBar()){
            sell ();
            debugPrintln ("sell");
            isSend = true;
        }
    }
}
```

JS2：

```
var isSend = false;
function OnInit() {
    EnableTrading();

    var intQuan = getDefaultQuantity();
    if(intQuan<1000){
        setDefaultQuantity(10000);
    }
}
function OnTick() {
    if (!isSend) {
        if (!isHistoryBar()) {
            sell ("FX01","EUR/USD");
            debugPrintln ("sell, enter all parameter");
            isSend=true;
        }
    }
}
```

4.11 sendOrder

Description:

- **sendOrder** *(sSide, sOrderType, iQuantity [,sSymbol] [,dStopPrice] [,dLimitPrice] [,dSL] [,dTP] [,sAccountName] [,sExpiry] [,sMMID] [, sFillType])*

Submit an order with all parameter to broker.

Parameters:

118. *sSide:* Required. A String value. The following constants are available:

Constant	Comment
buy	Submit a buy order.
sell	Submit a sell order.

119. *sOrderType:* Required. A String value. The following constants are available:

Constant	Comment
market	Submit a market order.
limit	Submit a limit order.
stop	Submit a stop order.
threshold	Submit a threshold order.

120. *iQuantity*: Required, a positive integer value. The number of shares/contracts.

121. *sSymbol:* Optional. The symbol for which to initiate the trade.

122. *dStopPrice:* Optional. See below: “stopPrice, limitPrice Explain”.

123. *dLimitPrice:* Optional. See below: “stopPrice, limitPrice Explain”.

124. *dSL:* Optional. Stop Loss.

125. *dTP:* Optional. Take Profit.

126. *sAccountName:* Optional. The account name for which to initiate the trade.

127. *sExpiry:* Optional. Expiration for the order. The following constants are available:

Constant	Comment
GTC	TIF Setting is “GTC”
IOC	TIF Setting is “IOC”
FOK	TIF Setting is “FOK”

128. *sMMID:* Optional.

129. *sFillType:* Optional. The following constants are available:

Constant	Comment
PART	default
AON	

dStopPrice, dLimitPrice Explain：

sOrderType	dStopPrice	dLimitPrice
market	Invalid	Invalid
stop	Stop Price	Invalid
limit	Invalid	Limit Price
“Threshold”	Lower limit	Upper limit

Example :

JS：

```
var isSend = false;
function OnInit() {
EnableTrading();
}
function OnTick() {
    if (!isSend) {
        if (!isHistoryBar()) {
            var upper = getQuote("bid") + 0.02;
            var lower = getQuote("bid") - 0.05;
            sendOrder ("sell","threshold",10000,"AUD/CAD",lower,upper,0,0);
            debugPrintln ("sell");
            isSend=true;
        }
    }
}
```

5. Account and Portfolio Functions

5.1 Account Object

Description:

Account Object includes some properties and methods about account information. There are two methods for obtaining **Account** object:

1. We can use **getCurrentAccount** function to get the Account object, as follows:

```
var account = getCurrentAccount ();
```

Note: This method only obtains the account object that is currently logged on Fortex.

2. We can use **new** method to create the Account object, as follows:

```
var acc= new Account (AccountID);
```

Note: AccountID is account name. This method can obtain any specified account object.

If the return value is 0 or a negative integer value in the Account Object, it means the function fails to retrieve the quote.

Function List:

Account object have the following functions:

NO,	Functions	Comment
1	**getAccountName** ()	Retrieve the Account Name
2	**getPosition** ()	Retrieve the amount of Position

3	**getOpenPL** ()	Retrieve the value of OpenPL
4	**getClosePL** ()	Retrieve the value of ClosePL
5	**getPL** ()	Retrieve the value of PL
6	**getQty** ()	Retrieve the value of QTY
7	**getTickets** ()	Retrieve the amount of Tickets
8	**getAvailableMargin** ()	Retrieve the value of Available_Margin
9	**getAssetBySymbolName**([*symbolName*])	Retrieve the Asset by symbol name

5.1.1 getAccountName

Description:

- **getAccountName ()**

Retrieve the Account Name.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
var objAcc = getCurrentAccount();
      if (objAcc!= null)   {
            debugPrintln ("account:" + objAcc.getAccountName());
      }
}
function OnTick(){
}
```

5.1.2 getValue

Description:

- **getValue()**

Retrieve the amount of Value.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objAcc = getCurrentAccount();
    if(objAcc!= null){
        debugPrintln ("Value:" + objAcc.getValue());
    }
}
function OnTick() {
}
```

5.1.3 getEquity

Description:

- **getEquity()**

Retrieve the amount of Equity.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objAcc = getCurrentAccount();
    if(objAcc!= null){
        debugPrintln ("Equity:" + objAcc.getEquity ());
    }
}
function OnTick() {
}
```

5.1.4 getBuyingPower

Description:

- **getEquity()**

Retrieve the amount of BuyingPower.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objAcc = getCurrentAccount();
    if(objAcc!= null){
        debugPrintln ("BuyingPower:" + objAcc.getBuying_Power ());
    }
}
function OnTick() {
}
```

5.1.5 getLeverage_Ratio

Description:

- **getLeverage_Ratio()**

Retrieve the valueof Leverage Ratio.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objAcc = getCurrentAccount();
    if(objAcc!= null){
        debugPrintln ("Leverage Ratio :" + objAcc.getLeverage_Ratio ());

    }
}
function OnTick() {
}
```

5.1.6 getRequiredMargin

Description:

- **getRequiredMargin()**

Retrieve the value of RequiredMargin.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objAcc = getCurrentAccount();
    if(objAcc!= null){
    debugPrintln ("RequiredMargin:" +  objAcc.getRequiredMargin());

    }
}
function OnTick() {
}
```

5.1.7 getMaintenanceMargin

Description:

- **getMaintenanceMargin ()**

Retrieve the value of MaintenanceMargin.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objAcc = getCurrentAccount();
    if(objAcc!= null){
```

```
        debugPrintln("MainenanceMargin:"+ objAcc.getMaintenanceMargin());
    }
    }
    function OnTick() {
    }
```

5.1.8 getLiquidationMargin

Description:

- **getLiquidationMargin ()**

Retrieve the value of LiquidationMargin.

Parameters:

None, function takes no parameters.

Example

JS：

```
    function OnInit() {
        var objAcc = getCurrentAccount();
        if(objAcc!= null){
        debugPrintln ("LiquidationMargin :" + objAcc.getLiquidationMargin());

        }
    }
    function OnTick() {
    }
```

5.1.9 getMargin_Ratio

Description:

- **getMarginRatio ()**

Retrieve the value of MarginRatio.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objAcc = getCurrentAccount();
    if(objAcc!= null){
    debugPrintln ("MarginRatio:" + objAcc.getMargin_Ratio());
    }
}
function OnTick() {
}
```

5.1.10 getCommission

Description:

- **getCommission()**

Retrieve the value of Commission.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objAcc = getCurrentAccount();
    if(objAcc!= null){
    debugPrintln ("Commission:" + objAcc.getCommission());
    }
}
function OnTick() {
}
```

5.1.11 getPosition

Description:

- **getPosition ()**

Retrieve the amount of Position.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objAcc = getCurrentAccount();
    if(objAcc!= null){
        debugPrintln ("position:" + objAcc.getPosition ());
    }
}
function OnTick() {
}
```

5.1.12 getOpenPL

Description:

- **getOpenPL ()**

Retrieve the value of OpenPL.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objAcc = getCurrentAccount();
    if (objAcc!= null)  {
        debugPrintln("open P&L:" + objAcc.getOpenPL());
    }
}
function OnTick() {
}
```

5.1.13 getClosePL

Description:

- **getClosePL ()**

Retrieve the value of ClosePL.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objAcc = getCurrentAccount ();
    if (objAcc!= null)  {
        debugPrintln ("close P&L:" + objAcc.getClosePL ());
    }
}
function OnTick(){
}
```

5.1.14 getPL

Description:

- **getPL ()**

Retrieve the value of PL.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objAcc = getCurrentAccount ();
    if (objAcc!= null)  {
        debugPrintln ("P&L:" + objAcc.getPL ());
```

```
        }
    }
    function OnTick(){
    }
```

5.1.15 getQty

Description:

- **getQty ()**

Retrieve the value of QTY.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objAcc = getCurrentAccount();
    if (objAcc!= null)  {
        debugPrintln ("QTY:" + objAcc.getQty ());
    }
}
function OnTick() {
}
```

5.1.16 getTickets

Description:

- **getTickets ()**

Retrieve the amount of Tickets.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objAcc = getCurrentAccount ();
    if (objAcc!= null)  {
        debugPrintln ("tickets:" + objAcc.getTickets ());
    }
}
function OnTick() {
}
```

5.1.17 getAvailableMargin

Description:

- **getAvailableMargin ()**

Retrieve the value of Available_Margin.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objAcc = getCurrentAccount();
    if (objAcc!= null)  {
        debugPrintln ("available_margin:" + objAcc.getAvailableMargin ());
    }
}
function OnTick(){
}
```

5.1.18 getAssetBySymbolName

Description:

- **getAssetBySymbolName** ([*symbolName*])

Retrieve the **Asset** object by symbol name.

Parameters:

130.*symbolName:* Optional. The symbol is the symbol of Portfolio window.

Example

JS：

```
function OnInit() {
}
function OnTick() {
    var objAcc = new Account("FX01");
    var objAsset = objAcc.getAssetBySymbolName ("USD/JPY");
    debugPrintln ("assert for FX01 symbol USD/JPY :" + objAsset.getPosition ());
}
```

5.2 Asset Object

Description:

Asset object includes some properties and methods about portfolio. There are two methods for obtaining asset object:

3. We can use the **getAssetBySymbolName** function to get Asset object, as follows:

```
var objAcc = new Account ("SA07");
var objAsset = objAcc.getAssetBySymbolName ("EUR/JPY");
```

4. We can use **new** method to create the Account object, as follows:

```
var objAsset = new Asset (AccountID, symbolName);
```

Note: AccountID is account name. This method can obtain any specified asset object. If the return value is 0 or a negative integer value in the Asset Object, it means the function fails to retrieve the quote.

Function List：

Asset object have the following functions:

NO.	Function	Description
1	**getSymbol** ()	Retrieve the symbol
2	**getPosition** ()	Retrieve the value of Position
3	**getPrice** ()	Retrieve the Price
4	**getOpenPL** ()	Retrieve the value of OpenPL
5	**getClosePL** ()	Retrieve the value of ClosePL
6	**getPL** ()	Retrieve the value of PL
7	**closePosition**(*trade*[, *quantity*] [, *expiry*])	Submit a close position operation

5.2.1 getSymbol

Description:

- **getSymbol ()**

Retrieve the symbol in the portfolio window.

Parameters:

None, function takes no parameters.

Example

```
function OnInit() {
    var objAcc = getCurrentAccount();
    var objAsset = objAcc.getAssetBySymbolName ();
    if (objAsset!= null){
        debugPrintln (" symbol:" + objAsset.getSymbol());
    }
}
function OnTick(){
}
```

5.2.2 getSide

Description:

- **getSide()**

Retrieve the value of Side (Short or Long) in the portfolio window.

Parameters:

None, function takes no parameters.

Example

JS:

```
function OnInit() {
    var objAcc = getCurrentAccount();
    var objAsset = objAcc.getAssetBySymbolName();
    if (objAsset!= null){
        debugPrintln("Side:" + objAsset.getSide());
```

```
        }
    }
    function OnTick(){
    }
```

5.2.3 getPosition

Description:

- **getPosition()**

Retrieve the value of Position in the portfolio window.

Parameters:

None, function takes no parameters.

Example

JS:

```
    function OnInit() {
        var objAcc = getCurrentAccount();
        var objAsset = objAcc.getAssetBySymbolName();
        if (objAsset!= null){
            debugPrintln("position:" + objAsset.getPosition());
        }
    }
    function OnTick(){
    }
```

5.2.4 getPrice

Description:

- **getPrice ()**

Retrieve the Price in the portfolio window.

Parameters:

None, function takes no parameters.

Example

JS:

```
function OnInit() {
    var objAcc= getCurrentAccount();
    var objAsset = objAcc.getAssetBySymbolName();
    if (objAsset!= null){
        debugPrintln ("price:" + objAsset.getPrice());
    }
}
function OnTick(){
}
```

5.2.5 getOpenPL

Description:

- **getOpenPL ()**

Retrieve the value of OpenPL in the portfolio window.

Parameters:

None, function takes no parameters.

Example

JS:

```
function OnInit() {
    var objAcc = getCurrentAccount();
    var objAsset = objAcc.getAssetBySymbolName();
    if (objAsset!= null){
        debugPrintln ("open P&L:" + objAsset.getOpenPL());
    }
}
function OnTick(){
}
```

5.2.6 getClosePL

Description:

- **getClosePL ()**

Retrieve the value of ClosePL in the portfolio window.

Parameters:

None, function takes no parameters.

Example

JS:

```
function OnInit() {
    var objAcc = getCurrentAccount();
    var objAsset = objAcc.getAssetBySymbolName();
    if (objAsset!= null){
        debugPrintln ("close P&L:" + objAsset.getClosePL());
    }
}
function OnTick(){
}
```

5.2.7 getPL

Description:

- **getPL ()**

Retrieve the value of PL in the portfolio window.

Parameters:

None, function takes no parameters.

Example

JS:

```
function OnInit() {
    var objAcc = getCurrentAccount();
    var objAsset = objAcc.getAssetBySymbolName();
    if (objAsset!= null){
        debugPrintln ("P&L:" + objAsset.getPL());
    }
}
function OnTick(){
}
```

5.2.8 closePosition

Description:

- **closePosition ()**

Submit a close position operation.

Parameters:

None, function takes no parameters.

Example

JS：

```
var flag = 0;
function OnInit(){
EnableTrading();

}
function OnTick(){
      var objAcc= getCurrentAccount();
      var objAsset = objAcc.getAssetBySymbolName("EUR/JPY");
            if (! isHistoryBar()){
                  if (flag == 0){
                        if (objAsset.getPosition() != null){
            objAsset.closePosition ();
            debugPrintln ("close position");
                                    }
            flag = 1;
                  }

      }
}
```

6. Order Manager Functions

6.1 Order Manager Object

Description:

OrderManager Object includes some properties and methods about order manager. It is not created in your script.

If the return value is 0 or a negative integer value in the Order Manager Object, it means the function fails to retrieve the quote.

Function List：

OrderManager object have the following functions:

NO.	Function	Description
1	OrderManager.**getOrders**([*sAccountName*] [,*sSymbol*])	Retrieve the object of Order
2	OrderManager.**getOpenOrders**([*sAccountName*] [,*sSymbol*])	Retrieve the Open Orders
3	OrderManager.**getFilledOrders**([*sAccountName*] [,*sSymbol*])	Retrieve the Filled Orders
4	OrderManager.**getCancelledOrders**([*sAccountName*] [,*sSymbol*])	Retrieve the Cancelled Orders
5	OrderManager.**cancelOrder** (*sOrderID*)	Cancel the pending order
6	OrderManager.**teplaceOrder***(sOrder*, *dStopPrice*, *dLimitPrice*, *iNewQuantity*, *sNewOrderType*)	Replace the pending order

1. OrderManager.getOrders

Description:

- **OrderManager.getOrders (**[*sAccountName*],[*sSymbol*]**)**

Retrieve the object of Order.

Parameters:

131.*sAccountName:* Optional. A String value.
132.*sSymbol:* Optional. A String value.

Example

JS：

```
function OnInit() {
    var objOrder= OrderManager.getOrders();
    debugPrintln ("The order's lengths is :"    +    objOrder.length);
    for (var i = 0; i < objOrder.length; i++){
debugPrintln ("Order" + i + " Acct:" + objOrder[i].getAccount());
        debugPrintln ("Order" + i + " Symbol:" + objOrder[i].getSymbol());
        debugPrintln ("Order" + i + " QTY:" + objOrder[i].getQty());
```

```
            debugPrintln ("Order" + i +" price:" + objOrder[i].getPrice());
            debugPrintln ("Order" + i + " stop price:" + objOrder[i].getStopPrice());
            debugPrintln ("Order" + i + " limit price:" + objOrder[i].getLimitPrice());
            debugPrintln ("Order" + i + " Type:" + objOrder[i].getType());
            debugPrintln ("Order" + i +" Status:" + objOrder[i].getStatus());
            debugPrintln ("Order" + i +" ID:" + objOrder[i].orderID);
            debugPrintln ("Order" + i +" Side:" + objOrder[i].getSide());
        }
    }
    function OnTick() {
    }
```

6.1.1 OrderManager.getOpenOrders

Description:

- **OrderManager.getOpenOrders** ([*sAccountName*] [,*sSymbol*])

Retrieve the Open Orders.

Parameters:

133. *sAccountName:* Optional. A String value.
134. *sSymbol:* Optional. A String value.

Example

JS:

```
    function OnInit() {
        var objOrders = OrderManager.getOpenOrders();
        debugPrintln ("The open orders' length is: " + objOrders.length);
        for (var i = 0; i < objOrders.length; i++){
            debugPrintln ("Order" + i + " Symbol:" + objOrders[i].getSymbol());
            debugPrintln ("Order" + i + " QTY:" + objOrders[i].getQty());
            debugPrintln ("Order" + i +" price:" + objOrders [i].getFilledPrice());
            debugPrintln ("Order" + i + " stop price:" + objOrders[i].getStopPrice());
            debugPrintln ("Order" + i + " limit price:" + objOrders[i].getLimitPrice());
            debugPrintln ("Order" + i + " Type:" + objOrders[i].getType());
            debugPrintln ("Order" + i +" Status:" + objOrders[i].getStatus());
            debugPrintln ("Order" + i +" ID:" + objOrders[i].orderID);
            debugPrintln ("Order" + i +" Side:" + objOrders[i].getSide());
            debugPrintln ("Order" + i +" SL:" + objOrders[i].getStop_Loss());
            debugPrintln ("Order" + i +" TP:" + objOrders[i].getTake_Profit());
            debugPrintln ("Order" + i +" date:" + objOrders[i].getTradeDate());
```

```
            debugPrintln ("Order" + i +" time:" + objOrders[i].getTradeTime());

        }
    }
    function OnTick() {
    }
```

6.1.2 OrderManager.getFilledOrders

Description:

- **OrderManager. getFilledOrders** ([*sAccountName*],[*sSymbol*])

Retrieve the Filled Orders.

Parameters:

135. *sAccountName:* Optional. A String value.
136. *sSymbol:* Optional. A String value.

Example

JS:

```
function OnInit() {
    var objOrder= OrderManager.getFilledOrders ();
    debugPrintln ("the filled orders' length is: "+ objOrder.length);
    for (var i=0;i< objOrder.length;i++){
        debugPrintln ("Order" + i + " Symbol:" + objOrder [i].getSymbol());
        debugPrintln("Order" + i + " QTY:" + objOrder [i].getQty());
        debugPrintln ("Order" + i +" price:" + objOrder [i].getFilledPrice());
        debugPrintln ("Order" + i + " stop price:" + objOrder [i].getStopPrice());
        debugPrintln ("Order" + i + " limit price:" + objOrder [i].getLimitPrice());
        debugPrintln ("Order" + i + " Type:" + objOrder [i].getType());
        debugPrintln ("Order" + i +" Status:" + objOrder [i].getStatus());
    }
}
function OnTick() {
}
```

6.1.3 OrderManager.getCancelledOrders

Description:

- **OrderManager.getCancelledOrders** ([*sAccountName*],[*sSymbol*])

Retrieve the Cancelled Orders.

Parameters:

137. *sAccountName:* Optional. A String value.
138. *sSymbol:* Optional. A String value.。

Example

JS：

```
function OnInit() {
    var objOrder = OrderManager.getCancelledOrders();
    debugPrintln ("The cancelled orders’ length is: " + objOrder.length);
    for(var i = 0; i < objOrder.length; i++){
        debugPrintln ("Order" + i + " Symbol is:" + objOrder [i].getSymbol());
        debugPrintln ("Order" + i + " QTY is:" + objOrder [i].getQty());
        debugPrintln ("Order" + i +" price is:" + objOrder [i].getPrice());
        debugPrintln ("Order" + i + " stop price:" + objOrder [i].getStopPrice());
        debugPrintln ("Order"+ i + " limit price is:" +
objOrder[i].getLimitPrice());
        debugPrintln ("Order" + i + " Type is:" + objOrder [i].getType());
        debugPrintln ("Order" + i +" Status is:" + objOrder [i].getStatus());
    }
}
function OnTick() {
}
```

6.1.4 OrderManager.cancelOrder

Description:

- **OrderManager. cancelOrder** (*sOrderID*)

Cancel the pending order.

Parameters:

139. *OrderID*: Required. The object of specified order.

Example

JS:

```
var flag = 0;
function OnInit(){
   EnableTrading();
}
function OnTick(){
      var obj_order;
      if (isHistoryBar()){return;}
      if (flag == 0){
            var price = getQuote("bid") - 0.03;
            OrderID = sellStop(16000, price);
            if (OrderID != null){
                  debugPrintln("Place a sell stop order,it is"+OrderID);
                  sleep(5000);
                  OrderManager.cancelOrder(OrderID);
                  debugPrintln("cancel order");
             }
             else{
                debugPrintln("Send Order Failed!");
                return;
             }
             flag = 1;
       }
}
```

6.1.5 OrderManager.replaceOrder

Description:

- **OrderManager.replaceOrder** (*sOrder*, *dStopPrice*, *dLimitPrice*, *iNewQuantity*, *sNewOrderType*)

Replace the pending order.

Parameters:

140. *sOrder*: Required. The ID of the specified order.
141. *dStopPrice / dLimitPrice*: Required. See below: “newPrice1, newPrice2 Explain”.
142. *iNewQuantity*: Required. A positive integer value.
143. *sNewOrderType*: Required. A String value. The following constants are available:

Constant	Comment
market	Submit a market order.
limit	Submit a limit order.
stop	Submit a stop order.
threshold	Submit a threshold order.

NewPrice1, NewPrice2 Explain:

newOrderType	newPrice1	newPrice2
market	0	0
stop	Stop Price	0
limit	0	Limit Price
Threshold	LowerPrice	Upper Price

Example

JS：

```
var IsOrderSended = false;
function OnInit() {
   EnableTrading();
}
function OnTick() {
      var objOrd;
      if (isHistoryBar())
            return;
      if (IsOrderSended == false) {
            var price =getQuote ("bid") - 0.02;
            var OrderID = sellStop (13000, price);
            if (OrderID!=null){
                  objOrd = new Order(OrderID);
                  debugPrintln ("A sell Stop is sended ");
            }
            else{
                  debugPrintln ("Send Order Failed!");
                  return;
            }
            sleep(6000)
price1 = getQuote ("ask") -0.05;
qty = objOrd.getQty () + 10000;
OrderID=OrderManager.replaceOrder (objOrd, price1, 0, qty,"Stop");
            debugPrintln("replace order");
            IsOrderSended = true;
      }
}
```

6.2 Order Object

Description:

Order Object includes some properties and methods about order. There are two

methods for obtaining asset object:

5. We can use the **GetOrders** function to get the order object, as follows:

```
var objOrder= OrderManager.getOrders ();
```

6. We can use the **new** method to create the order object, as follows:

```
var objOrder = new Order (orderID);;
```

Note: ordered is the ID of the specified order。
If the return value is 0 or a negative integer value in the Order Object, it means the function fails to retrieve the quote.

Function List：

Order object have the following functions:

NO.	Function	Description
1	Order.**getSymbol** ()	Retrieve the Symbol of pending order
2	Order.**getQty** ()	Retrieve the value of QTY
3	Order.**getPrice** ()	Retrieve the price of Order
4	Order.**getStopPrice** ()	Retrieve the StopPrice of Order
5	Order.**getLimitPrice** ()	Retrieve the LimitPrice of Order
6	Order.**getType** ()	Retrieve the orderType of Order
7	Order.**getMarketable** ()	Retrieve the properties of Order
8	Order.**getStatus** ()	Retrieve the status of Order
9	Order.**getRoute** ()	Retrieve the Route of Order
10	Order.**cancel** ()	Submit a cancel order operation

6.2.1 Order.getSymbol

Description:

- **Order.getSymbol ()**

Retrieve the Symbol of pending order.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objOrder = new Order("69673");
    debugPrintln ("The symbol name is: "+ objOrder. getSymbol());
```

```
}

function OnTick(){
}
```

6.2.2 Order.getQty

Description:

- **Order.getQty ()**

Retrieve the value of QTY.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objOrder = new Order ("69673");
    debugPrintln ("The Order Qty is:"+ objOrder. getQty ());
}
function OnTick(){
}
```

6.2.3 Order.getFilledPrice

Description:

- **Order.getFilledPrice()**

Retrieve the Market price of filled Order.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objOrder = new Order ("69673");
    debugPrintln ("The Order Quote is:"+ objOrder. getFilledPrice());
}
function OnTick(){
}
```

6.2.4 Order.getStopPrice

Description:

- **Order. getStopPrice ()**

Retrieve the StopPrice of Order.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objOrder = new Order("69676");
    debugPrintln ("The Order Stop Price is:"+ objOrder. getStopPrice ());
}
function OnTick(){
}
```

6.2.5 Order.getLimitPrice

Description:

- **Order. getLimitPrice ()**

Retrieve the LimitPrice of Order.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objOrder = new Order("69675");
    debugPrintln ("The Order Limit Price is "+ objOrder. getLimitPrice ());
}
function OnTick(){
}
```

6.2.6 Order.getType

Description:

- **Order. getType ()**

Retrieve the orderType of Order.

Parameters:

None, function takes no parameters.

Example

JS:

```
function OnInit() {
    var objOrder = new Order ("69676");
    debugPrintln ("The OrderType is:"+ objOrder. getType ());
}
function OnTick() {
}
```

6.2.7 Order.getMarketable

Description:

- **Order. getMarketable ()**

Retrieve the properties of Order.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objOrder = new Order ("69676");
    debugPrintln ("The Order Marketable is:"+ objOrder. getMarketable ());
}
function OnTick() {
}
```

6.2.8 Order.getStatus

Description:

- **Order. getStatus ()**

Retrieve the status of Order.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objOrder = new Order("69676");
    debugPrintln ("The Order Status is:"+ objOrder. getStatus ());
}
function OnTick() {
}
```

6.2.9 Order.getRoute

Description:

- **Order. getRoute ()**

Retrieve the Route of Order.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objOrder = new Order (“69696”);
    debugPrintln ("The Order Route is:"+ objOrder. getRoute ());
}
function OnTick() {
}
```

6.2.10 Order.getSide

Description:

- **Order. getSide ()**

Retrieve the Side of Order.

Parameters:

None, function takes no parameters.

Example

```
function OnInit() {
    var objOrder = new Order("67919");
    debugPrintln ("The Order Side is: "+ objOrder. getSide ());
}
function OnTick() {
}
```

6.2.11 Order.getTradeDate

Description:

- **Order. getTradeDate()**

Retrieve the TradeDate of Order.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objOrder = new Order("69695");
    debugPrintln ("The Order TradeDate is: "+ objOrder. getTradeDate ());
}
function OnTick() {
}
```

6.2.12 Order.getTradeTime

Description:

- **Order. getTradeTime()**

Retrieve the TradeTime of Order.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objOrder = new Order("69695");
    debugPrintln ("The Order TradeTime is: "+ objOrder. getTradeTime());
}
function OnTick() {
}
```

6.2.13 Order.getStop_Loss

Description:

- **Order.getStop_Loss()**

Retrieve the StopLoss Price of Order.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objOrder = new Order("69696");
    debugPrintln ("The Stop_Loss Price is: "+ objOrder. getStop_Loss());
}
function OnTick() {
}
```

6.2.14 Order.getTake_Profit

Description:

- **Order.getTake_Profit()**

Retrieve the Take_Profit Price of Order.

Parameters:

None, function takes no parameters.

Example

JS：

```
function OnInit() {
    var objOrder = new Order("69696");
    debugPrintln ("The Stop_Loss Price is: "+ objOrder. getTake_Profit());
}
function OnTick() {
}
```

6.2.15 Order.cancel

Description:

- **Order. cancel ()**

Submit a cancel order operation.

Parameters:

None, function takes no parameters.

Example

JS:

```
var isOrderSended = false;
function OnInit() {
EnableTrading();
}
function OnTick() {
      var objOrder;
      if (isHistoryBar())
            return;
 if (isOrderSended == false) {
            var price = getQuote ("bid") - 0.03;
            var OrderID = sellStop (12000, price);
            if (OrderID!=null) {
                  objOrder = new Order(OrderID);
                  debugPrintln("A sell Stop is sended");
                        sleep(3000);
            }
            else{
                  debugPrintln("Send Order Failed!");
                  return;

            }
            str_OrderID = objOrder.cancel();
            debugPrintln ("cancel order");
            isOrderSended = true;
      }
}
```

7. Utility Functions

7.1 FunctionParameter Object

Description:

- **FunctionParameter (***paramString***, *ParamType*)**

FunctionParameter Object class is a set of objects and functions that allow you to

manage your input parameters in a script.

Parameters:

144. *paramString*: Required. the name of the parameter as defined in OnTick()
145. *paramType*: Required. The type of parameter object. See Parameter Types below. The following constants are available:

Constant	Comment
FunctionParameter.**STRING**	A string menu object. either a single string or a list of string options
FunctionParameter.**NUMBER**	A numeric menu object
FunctionParameter.**COLOR**	A color menu object
FunctionParameter.**BOOLEAN**	A Boolean menu object

FunctionParameter Methods:

Method	Comment
setName()	Set the display name of this menu item. This allows you to create a more descriptive name to display to the user
addOption()	Add a string option
setLowerLimit()	Set the lowest acceptable input value for a numeric menu option
setUpperLimit()	Set the highest acceptable input value for a numeric menu option
setDefault()	Set the default value for this menu option

Example

```
var arrFP = new Array ();
function OnInit() {
    var x;
     //initialize formula parameters
     x = 0;
    //define a numerical menu option
    arrFP[x] = new FunctionParameter ("Pa1", FunctionParameter.NUMBER);
    with (arrFP[x]) {
        setName("Numeric Menu Option");
        setLowerLimit (5);
        setUpperLimit (125);
        setDefault (40);
    }
    x++;
    //define a string list menu option
    arrFP[x] = new FunctionParameter ("Pa2", FunctionParameter.STRING);
    with (arrFP[x]) {
        setName ("String Menu Option");
```

```
            addOption("String1");
            addOption("String2");
            setDefault("String1");
        }
        x++;
        //define a color menu option
        arrFP[x] = new FunctionParameter ("Pa3", FunctionParameter.COLOR);
        with (arrFP[x]) {
            setName( "Color Menu Option" );
            setDefault( Color.blue );
        }
        x++;
        //define a Boolean menu option
        arrFP[x] =new FunctionParameter("Pa4", FunctionParameter.BOOLEAN);
        with ( arrFP[x] ) {
            setName("Bool Menu Option");
            setDefault (true);
        }
    }
    //make sure that our 4 menu parameters are included in OnTick's declaration
    function OnTick( Pa1, Pa2, Pa3, Pa4 ) {
    }
```

7.2 call

Description:

- **call (***efsname* [**,***arguments*]**)**

This function is provided for backwards compatibility with EFS1. It allows you to call another script and retrieve the return values from that script.

Parameters:

146. *efsname*: Required. A String value. The path and filename of the script to be called.
147. *arguments:* Optional. A String value. List of input parameters, if any, required by the script being called

Example

```
function OnTick() {
    var myVar;
     ...
     ...
```

```
        //call a second script that calculates a custom moving average
        //and retrieves the current value. In this example, the script
        //we are calling has one input parameter.
        myVar = call ( "myCustomMA.efs", 10 );
}
```

7.3 internalCall

Description:

- **internalcall (***functionName* [, *parameters*...])

Returns a series object representing the values calculated from a given function name. The function specified will be executed as if it were a OnTick() function being called in an external EFS.

Parameters:

148.*functionName*: Required. A String value. Name of user-defined function to call.

149.*parameters:* Optional. A String value. Parameters, if any, required by the user-defined function you are calling.

Note:

If you pass a series object to the **internalCalll** () function, be sure to pass the series as the last parameter. This will force **internalCall** () to execute in the series' sym/inv context.

Example

```
function OnInit() {
}
var IsInit = false;
var callResult;

function OnTick() {
      if (IsInit == false) {
            callResult = internalCall (IamOpenAndClose);
      }
      return callResult.getValue();
}

function IamOpenAndClose() {
    return (open() + close()) / 2;
}
```

7.4 ref

Description:

- **ref (*barIndex*)**

The **ref**() function allows you to retrieve prior values for a built-in study (or studies). If multiple return values are being used then the **ref**() function will return its results in an array.

Parameters:

150. *barIndex*: Required. Offset indicating which historical indicator data points to retrieve

Example

```
function OnInit() {
}
function OnTick() {
    var RefValue = ref (-1);
    var thisValue = open();
    debugPrintln ("RefValue " + RefValue);
    debugPrintln ("thisValue " + thisValue);
    return thisValue;
}
```

8. Quote Object

Description:

The **Quote** Object class is a set of field that allows you to store the quote information in script.

8.1 Quote info

We can use getQuoteInfo function to get the last valid Quote object:

```
var objOrder= getQuoteInfo();
```

Attribute List:

The object contains 15 field values, show below:

NO.	Attribute	Comment
1	**Symbol**	Retrieve the name of the symbol currently being

		charted
2	**Bid**	Retrieve the bid price
3	**Ask**	Retrieve the ask price
4	**BidSize**	Retrieve current symbol's bid size
5	**AskSize**	Retrieve current symbol's ask size
6	**BidMMID**	bidExchange of L1
7	**AskMMID**	askExchange of L1
8	**Spread**	Retrieve the value of spread
9	**QuoteTick**	Tick for quote price

Example：

JS:

```
function OnInit(){
    }
function OnTick(){
    if (isHistoryBar()){
return;
}
var quote = getQuoteInfo();
debugPrintln ("sym:" + quote.Symbol);
debugPrintln ("bid:" + quote.Bid);
debugPrintln ("ask:" + quote.Ask);
debugPrintln ("bidsize:" + quote.BidSize);
debugPrintln ("asksize:" + quote.AskSize);
debugPrintln ("bidmmid:" + quote.BidMMID);
debugPrintln ("askmmid:" + quote.AskMMID);
debugPrintln ("spread:" + quote.Spread);
debugPrintln ("quotetick:" + quote.QuoteTick);
}
```

9. Appendix

9.1 Available Colors

Constant	Comment
Color.**white**	*Sample*
Color.**black**	*Sample*
Color.**darkgrey**	*Sample*
Color.**grey**	*Sample*
Color.**lightgrey**	*Sample*
Color.**navy**	*Sample*

Color.**blue**	*Sample*
Color.**aqua**	*Sample*
Color.**cyan**	*Sample*
Color.**teal**	*Sample*
Color.**darkgreen**	*Sample*
Color.**green**	*Sample*
Color.**lime**	*Sample*
Color.**olive**	*Sample*
Color.**khaki**	*Sample*
Color.**brown**	*Sample*
Color.**purple**	*Sample*
Color.**red**	*Sample*
Color.**magenta**	*Sample*
Color.**maroon**	*Sample*
Color.**yellow**	*Sample*
Color.**lightyellow**	*Sample*
Color.**paleyellow**	*Sample*