Table of Contents

Installing the ameLib Custom Library

When you “unzipped” the file that you downloaded containing the ameLib Custom Library files and this document that you are reading now, it should have created a folder named “experts”. To install the Library, simply copy the “experts” folder directly into your MetaTrader program folder. Make sure that MetaTrader is not running when you do this.

NOTE: If you have installed the AME Cross Trader, then it is not necessary to install the ameLib Custom Library as it is already part of the AME Cross Trader.

About the ameLib Custom Library

The ameLib Custom Library contains a small set of convenience functions to facilitate writing Expert Advisors. The Library comes complete with a Custom Template that automates some of the more tedious parts of writing Expert Advisors.

Trading functions

ame_OrderSend()	- Send order to the server. Same syntax as OrderSend
ame_OrderModify()	- Modify existing order. Same syntax as OrderModify
ame_OrderCloseAll()	- Close all orders opened by this Expert Advisor
ame_FirstOrder()	- Return the ID of the first open order in the list of open orders
ame_NextOrder()	- Return the ID of the next open order in the list of orders
ame_PipsGained()	- Return the number of pips gained or lost in a previously closed trade

Information functions

ame_AnyOpenOrders()	- Returns true if any orders opened by this EA are open
ame_CanAffordOrder()	- Returns true if there is enough equity to place an order
ame_FixLots()	- Returns a normalized lot size to pass to the ordering functions
ame_IsNewBar()	- Returns true if a new bar has formed in the last 60 seconds
ame_MagicNumber()	- Returns a magic number to be used by this EA for tracking orders
ame_OpenOrderCount()	- Returns the number of open orders opened by this EA

Persistence functions

ame_GetValue()	- Returns the value of a previously stored persistent variable
ame_SetValue()	- Sets or resets a persistent variable

Trailing Stop functions

ame_BeginTrailingStop() - Begins a monetary or percentage trailing stop

ame_CheckTrailingStop() - Returns true if the trailing stop has been reached

ame_ResetTrailingStop() - Resets the equity limit for a trailing stop

Miscellaneous functions

ame_CloseLibrary() - Save all persistent variables and close the Library

ame_InitLibrary() - Load all saved persistent variables and initialize the Library

ame_SetSendEmail() - Enable or disable sending of email when orders are placed

Creating an Expert Advisor that uses the Library

The easiest way to use the ameLib Custom Library is to create an Expert Advisor from the "AME Expert" Predefined Template, which comes with the library when you download it. To create a new Expert Advisor, simply select "New" from the "File" menu in the MetaEditor, then choose "Generate from template", then choose "Expert – AME Expert" and click "Next". On the next screen, fill in the name and add your parameters, then click "Finish". This will create a base EA that loads and unloads the library, and provides almost all of the functionality that you need to build an EA.

Figure 1. Creating a new Expert Advisor from template.

Modifying your newly created Expert Advisor

Once you have created a new Expert Advisor from the AME Expert Template, it needs a few modifications before it can be run on a live chart. These are described below.

placeBuyOrder() and placeSellOrder()

These two functions must be modified. Near the end of the functions you will find a line with a call to ame_OrderSend(). These are commented out and must be uncommented in order for the EA to place trades. These commands were placed in comments to prevent the EA from accidentally placing orders before it was modified. Also, there are comments inside of those functions describing how you can customize the functions if necessary. For example, changing the lot size of trades that it places.

Even if you want to create an EA from scratch, you should at least one time create one from the Template. Then you can study how the Library is included in the EA and initialized and de-initialized. These things are critical in the creation of an EA that uses the Library.

findEntrySignal()

This function is intentionally blank and must contain the logic that decides whether it's time to place a trade or not. The function gets called every tick and must return 1 for a buy signal, 2 for a sell signal, or 0 for no signal.

FindExitSignal()

This function must handle modifying of stop loss or take profit on open orders. It also can be used to decide if all trades must be closed. It must return true for a “panic” signal that forces all open trades to be closed, or return false if nothing else needs to happen.

Function Reference

ame_AnyOpenOrders

Returns true if any orders opened by this EA are still open.

bool **ame_AnyOpenOrders**();

Example:

```
if(ame_AnyOrdersOpen()) {
  ame_OrderCloseAll(slippage);
}
```

---

ame_BeginTrailingStop

Begins a trailing stop that is triggered when drawdown reaches the amount or percentage specified. Use this function in your init() function if you want to have a "panic mode" in your EA. Or use it when you place a trade along with ame_ResetTrailingStop().

void **ame_BeginTrailingStop**(

- double **dollars** - Amount of money or percentage loss to allow before a "stop" signal is triggered.

);

Example:

```
void init() {
  ame_InitLibrary(WindowExpertName());
  ame_BeginTrailingStop(0.4); // 40% max drawdown
}
```

---

ame_BeginTransaction

Begins a transaction and locks all other charts (who use transactions) from doing anything during that time. Returns true if the transaction lock can be secured. Returns false if the timeout period elapses while waiting for transaction lock.

void **ame_BeginTransaction**(

- string **mutexName** - Name of global variable to use for synchronizing.
- int **timeout** - Time in milliseconds to wait before timing out.
- string **comment** - Comment to output in Expert Log at beginning and end of transaction. Leave blank to supress output.

);

Example:

```
void placeGridOrders() {
  if(ame_BeginTransaction("_my_ea_name_", 30000, "place grid orders")) {
  {
    for(i=0; i<10; i++) {
      ame_OrderSend(...);
    }
    ame_EndTransaction("_my_ea_name_");
  }
}
```

---

ame_CanAffordOrder

Returns true if an order of a specified size can be placed with a specified amount of equity.

bool **ame_CanAffordOrder**(

- string **symbol** - Symbol name for the currency
- double **equity** - Amount of equity to use for the calculation
- double **lots** - Size of trade to use for the calculation

);

Example:

```
if(ame_CanAffordOrder(Symbol(), AccountEquity(), 1.0)) {
  orderId = ame_OrderSend(...);
}
```

---

ame_CheckTrailingStop

Returns true if drawdown has reached the "stopDollars" level.

bool **ame_CheckTrailingStop**();

Example:

```
if(ame_CheckTrailingStop()) {
  ame_OrderCloseAll(slippage);
}
```

---

ame_CloseLibrary

This function must be called in the deinit() function. It saves all persistent variables to the hard drive and frees up resources used by the EA.

bool **ame_CloseLibrary**();

Example:
```
void deinit() {
  ame_CloseLibrary();
  return(0);
}
```

---

ame_EndTransaction

Releases the transaction lock to allow other charts or threads to obtain a lock for themselves. If you call ame_BeginTransaction successfully, then at some time you MUST call ame_EndTransaction in order to allow other charts or threads to do things within a transaction.

bool **ame_EndTransaction**(string mutexName);

- string **mutexName** - name of global variable to use for synchronizing.

Example:
```
void placeGridOrders() {
  if(ame_BeginTransaction("_my_ea_name_", 30000, "place grid orders")) {
  {
    for(i=0; i<10; i++) {
      ame_OrderSend(...);
    }
    ame_EndTransaction("_my_ea_name_");
  }
}
```

---

ame_FirstOrder

The ameLib Custom Library keeps track of orders placed by the EA. Use this function to select the first in the list of open orders.

int **ame_FirstOrder**();

Example:
```
orderId = ame_FirstOrder();
if(orderId > 0) {
  OrderSelect(orderId, SELECT_BY_TICKET);
  ame_OrderModify(...);
}
```

---

ame_FixLots

Adjusts a value to conform to the number of digits allowed for placing a trade, setting stop loss or

setting take-profit.

double **ame_FixLots**(
- string **symbol** - Symbol to use for calculating precision
- double **lotSize** - Trade size to adjust based on the symbol passed

);

Example:
```
double lotSize = ame_FixLots(Symbol(), AccountEquity() / 10000.0);
```

---

ame_GetValue

Retrieves a value from the persistent variable array. The persistent variable array gets loaded from the hard drive when the EA starts up, and gets saved back when the EA shuts down.

double **ame_GetValue**(
- string **name** – The name of the variable to retrieve a value for.

);

Example:
```
double maxEquity = ame_GetValue("maxEquity");
if(AccountEquity() > maxEquity) {
  ame_SetValue("maxEquity", AccountEquity());
}
```

---

ame_InitLibrary

Initializes the library and loads the persistent variable array. This function must be called in your init() function in order for the ameLib Custom Library to work properly. Also, see ame_CloseLibrary().

bool **ame_InitLibrary**(
- string **eaName** - The name of the currently running EA. For best results, use WindowExpertName().

);

Example:
```
void init() {
  ame_InitLibrary(WindowExpertName());
}
```

---

ame_IsNewBar

Returns true if the time is within 1 minute of the open of a new bar. Not recommended for timeframes below H1.

bool **ame_IsNewBar**();

Example:

```
if((ame_AnyOrdersOpen() == false)
  && (ame_IsNewBar() == true)) {
  lookForTradeOpportunity();
}
```

---

ame_MagicNumber

Returns an automatically generated Magic Number for the chart that the EA is running on. The same Magic Number persists even if the EA has been shut down and restarted. It is this Magic Number that the Library uses to track which orders were placed by the EA for a given chart. It must be used in every call to ame_OrderSend() for proper tracking of orders to occur.

int **ame_MagicNumber**();

Example:

```
int ticket = ame_OrderSend(Symbol(), OP_BUY, 1, Ask, 3, Ask-25*Point,
                           Ask+25*Point, "My order #2", ame_MagicNumber(),
                           0, Green);
```

---

ame_NextOrder

Returns the Order ID of the next open order in the list of open orders that was opened by this EA on this chart.

int **ame_NextOrder**();

Example:

```
orderId = ame_FirstOrder();
while(orderId > 0) {
  OrderSelect(orderId, SELECT_BY_TICKET);
  ame_OrderModify(...);
  orderId = ame_NextOrder();
}
```

---

ame_OpenOrderCount

Returns the number of open orders that were opened by this EA on this chart.

int **ame_OpenOrderCount**();

Example:
```
int i, n = ame_OpenOrderCount();
for(i=0; i<n; i++) {
  doSomeThingHere();
}
```

---

ame_OrderCloseAll

Closes all open orders that were opened by this EA on this chart.

bool **ame_OrderCloseAll**(
- int **slippage** - Slippage in pips to allow when closing the order.

);

Example:
```
if(exitSignalFound()) {
  ame_OrderCloseAll(slippage);
}
```

---

ame_OrderModify

A reliable version of OrderModify, which handles retries when errors occur and waits if necessary for the server to allow trades.

int **ame_OrderModify**(
- int **ticket** - Unique number of the order ticket.
- double **price** - New open price of the pending order.
- double **stoploss** - New StopLoss level.
- double **takeprofit** - New TakeProfit level.
- datetime **expiration** - Pending order expiration time.
- color **arrow_color** - Arrow color for StopLoss/TakeProfit modifications in the chart or CLR_NONE to not display any arrow.

);

Example:
```
int orderId = ame_FirstOrder();
while(orderId > 0) {
  OrderSelect(orderId, SELECT_BY_TICKET);
```

```
  ame_OrderModify(orderId, OrderOpenPrice(), OrderStopLoss()-20*Point,
                  OrderTakeProfit()+20*Point, OrderExpiration(), CLR_NONE);
  orderId = ame_NextOrder();
}
```

---

ame_OrderSend

A reliable version of OrderModify, which handles retries when errors occur and waits if necessary for the server to allow trades.

int **ame_OrderSend**(

- string **symbol** - Symbol for trading.
- int **cmd** - Operation type. It can be any of the Trade operation enumeration.
- double **volume** - Number of lots.
- double **price** - Preferred price of the trade.
- int **slippage** - Maximum price slippage for buy or sell orders.
- double **stoploss** - Stop loss level.
- double **takeprofit** - Take profit level.
- string **comment** - Order comment text. Last part of the comment may be changed by server.
- int **magic** - Order magic number. May be used as user defined identifier.
- datetime **expiration** - Order expiration time (for pending orders only).
- color **arrow_color** - Color of the opening arrow on the chart or CLR_NONE to not display any arrow.

);

Example:

```
int ticket = ame_OrderSend(Symbol(), OP_BUY, volume, price, slippage, stoploss,
                           takeprofit, comment, ame_MagicNumber(), 0, Green);
```

---

ame_PipsGained

Returns the number of pips gained or lost for a given position, either open or closed.

int **ame_PipsGained**(

- int **orderID** - ID (ticket) of the order to check.

);

Example:

```
int orderId = ame_FirstOrder();
if(ame_PipsGained(orderId) < -50) {
  OrderClose(orderId, ...);
}
```

---

ame_ResetTrailingStop

Sets the equity monitor to zero, resetting the monetary trailing stop. Good for use when the monetary trailing stop is not used to halt all trading.

void **ame_ResetTrailingStop**();

Example:
```
if(ame_CheckTrailingStop()) {
  ame_OrderCloseAll(slippage);
  ame_ResetTrailingStop();
}
```

---

ame_SetValue

Stores a value in the persistent variable array. Each element in the persistent variable array has a name that must be used to reference that element. Also, the entire persistent variable array gets saved to the hard drive when ame_CloseLibrary() is called.

double **ame_SetValue**(
- string **name** - Name of the element in the array to access.
- double **value** - Value to store in the selected element.

);

Example:
```
double maxEquity = ame_GetValue("maxEquity");
if(AccountEquity() > maxEquity) {
  ame_SetValue("maxEquity", AccountEquity());
}
```

---

ame_SetSendEmail

Sets a flag that tells ame_OrderSend() whether or not to send email whenever it is called.

void **ame_SetSendEmail**(
- bool **state** - True to enable sending of emails, otherwise false.

);

Example:
```
void init() {
  ame_InitLibrary(WindowExpertName());
  ame_SetSendEmail(true);
}
```