Hedging Addendum

For the First Edition of *Expert Advisor Programming for MetaTrader 5*

In March 2016, MetaQuotes updated the MetaTrader 5 platform to allow hedging of positions, similar to MetaTrader 4. The changes to the platform and the language are described in this MQL5.com article:

https://www.mql5.com/en/articles/2299

The source code as described in the first edition of this book will not work on hedging accounts, as it was written for "netting" accounts. The second edition has been updated to include material on writing expert advisors for hedging accounts. If you own the first edition and have already developed trading systems for netting accounts, this document describes the differences in the downloadable source code for the second edition, as well as addressing the changes made to MQL5 for hedging accounts.

We've added a new include file, `TradeHedge.mqh`, that contains trading classes and functions that will work on hedging accounts. In addition, we've added hedging versions of all of the example expert advisors that were covered in the book.

The `TradeHedge.mqh` file contains several new classes: `CTradeHedge`, used to place and manage trades on hedging accounts, and `CPositions`, used to get order tickets and order counts for hedging orders. The standalone position information functions in the `Trade.mqh` file (described in Chapter 12 of the book) have been overridden by identically-named functions in the `TradeHedge.mqh` file that will work with hedging accounts.

To use the include file containing the hedging order classes, add this include directive to the top of your MQ5 file:

```
#include <Mql5Book\TradeHedge.mqh>
CTradeHedge Trade;
CPositions Positions;
```

Remove any references to the `Mql5Book\Trade.mqh` file, as it is already included.

The `CTradeHedge` class extends the `CTrade` class included in the `Trade.mqh` file, and overrides some of its functionality. All of the other functions in the `CTrade` class are still accessible to your program. The `Trade.mqh` contains a few changes – mainly bug fixes from the first edition, and to allow the `TradeHedge.mqh` file to inherit the private functions and objects from the `CTrade` class.

The `CPositions` class is a new class that allows easy retrieval of open order tickets, as well as a running count of currently open buy and sell orders on hedging accounts.

MQL5 Language Changes

Here are the major changes to the MQL5 language that were added to support hedging accounts:

MqlTradeRequest

The `MqlTradeRequest` class has a few fields added for modifying and closing hedging orders:

- `position` – The ticket number of a position to modify or close.
- `position_by` – The ticket number of an opposite position to close an order by. Used by the *Close By* operation.

When placing a hedging order, you do not need to assign anything to the `position` or `position_by` fields. It is strongly recommended that you assign a *magic number*, though:

```
request.action = TRADE_ACTION_DEAL;
request.type = ORDER_TYPE_BUY;
request.symbol = _Symbol;
request.magic = 123;
request.volume = TradeVolume;
request.price = SymbolInfoDouble(_Symbol,SYMBOL_ASK);
request.deviation = 50;

bool sent = OrderSend(request,result);
```

You can set the magic number in your expert advisor using an input variable that you'll pass to the `CTrade::MagicNumber()` function. It is recommended that you set this is the `OnInit()` event handler:

```
// Input variables
input ulong MagicNumber = 123;

// OnInit() event handler
int OnInit()
{
   Trade.MagicNumber(MagicNumber);
   return(0);
}
```

We'll use the magic number when retrieving order tickets using the functions in our `CPositions` class below.

When modifying or closing a hedging order, you'll need to assign the order ticket number to the `position` field:

```
request.action = TRADE_ACTION_DEAL;
request.type = ORDER_TYPE_BUY;
```

```
request.symbol = _Symbol;
request.position = sellTicket;
request.volume = PositionGetDouble(POSITION_VOLUME);
request.price = SymbolInfoDouble(_Symbol,SYMBOL_ASK);
request.deviation = 50;

bool sent = OrderSend(request, result);
```

The example above closes a sell order on a hedging account. We'll assume the `sellTicket` variable holds a valid open ticket number.

The `position_by` field is used for the *Close By* operation, which closes two opposite orders at the same time:

```
request.action = TRADE_ACTION_CLOSE_BY;
request.position = sellTicket;
request.position_by = oppositeBuyTicket;

bool sent = OrderSend(request, result);
```

PositionSelectByTicket()

For netting accounts, the `PositionSelect()` function selects a position by symbol for retrieving further information, using the `PositionGet...()` functions. For hedging accounts, the `PositionSelectByTicket()` function was added to do the same for hedging orders. It takes an order ticket number.

The example below selects an open order and retrieves the order type, assigning it to the `orderType` variable. We assume the `ticket` variable contains a valid order ticket:

```
PositionSelectByTicket(ticket);
ENUM_POSITION_TYPE orderType = PositionGetInteger(POSITION_TYPE);
```

PositionGetTicket()

The `PositionGetTicket()` function selects an order by its index in the order pool. The most common use of this function is to use a `for` loop with `PositionsTotal()` to loop through the open order pool. The example below gets a count of open orders:

```
ulong magicNumber = 123;
int buyCount, sellCount;

for(int i = 0; i < PositionsTotal(); i++)
{
   ulong ticket = PositionGetTicket(i);
   PositionSelectByTicket(ticket);
```

```
      if(PositionGetInteger(POSITION_MAGIC) != magicNumber && magicNumber > 0)
         continue;

      if(PositionGetInteger(POSITION_TYPE) == POSITION_TYPE_BUY)
      {
         buyCount++;
      }
      else if(PositionGetInteger(POSITION_TYPE) == POSITION_TYPE_SELL)
      {
         sellCount++;
      }
   }
```

The `PositionsTotal()` function determines how many orders are currently open, and also determines the exit condition for our `for` loop. We select each order in the order pool using `PositionGetTicket()`, passing in the zero-based index variable `i`. Then we use `PositionSelectByTicket()` to select the order for retrieving the order information. We then retrieve the magic number of the order and compare it against our `magicNumber` variable. If it does not match, we skip it. Otherwise, we get the order type and increment the `buyCount` or `sellCount` variable accordingly.

The CTradeHedge Class

The `CTradeHedge` class contains functions for placing, modifying and closing orders on hedging accounts. If you declare an object of the `CTradeHedge` class using the object name specified in the source code, you'll access all of these functions through the `Trade` object.

Buy() and Sell()

```
ulong Buy(string pSymbol, double pVolume, double pStop = 0,
      double pProfit = 0, string pComment = NULL);
ulong Sell(string pSymbol, double pVolume, double pStop = 0,
      double pProfit = 0, string pComment = NULL);
```

The `Buy()` and `Sell()` functions in the `CTradeHedge` class override the `Buy()` and `Sell()` functions in the `CTrade` class. They work identically to the functions in the `CTrade` class – the only difference is that they return the order ticket number of the order that was just placed.

ModifyPosition()

```
bool ModifyPosition(ulong pTicket, double pStop, double pProfit = 0);
```

The `ModifyPosition()` function takes the ticket number of a hedging order using the `pTicket` parameter, and modifies the stop loss or take profit of that order. It returns `true` if the order modification was successful.

Close()

```
bool Close(ulong pTicket, double pVolume = 0, string pComment = NULL);
```

The `Close()` function takes a ticket number of a hedging order using the `pTicket` parameter, and closes that order. It returns `true` if the order was closed successfully.

The CPositions Class

The `CPositions` class is used to retrieve a count of open orders by type, and to retrieve an array of ticket numbers. You will need the order ticket numbers to close hedging orders. If you declare an object of the `CPositions` class using the object name specified in the source code, you'll access these functions through the `Positions` object.

Get Order Counts

```
int Buy(ulong pMagicNumber);
int Sell(ulong pMagicNumber);
int TotalPositions(ulong pMagicNumber);
```

The `Buy()`, `Sell()` and `TotalPositions()` functions of the `CPositions` class will return a count of open orders of each type. The `TotalPositions()` function returns a count of all open orders, buy and sell.

Get Order Tickets

```
void GetBuyTickets(ulong pMagicNumber, ulong &pTickets[]);
void GetSellTickets(ulong pMagicNumber, ulong &pTickets[]);
void GetTickets(ulong pMagicNumber, ulong &pTickets[]);
```

The `GetBuyTickets()`, `GetSellTickets()` and `GetTickets()` functions retrieve an array of order tickets by order type. `GetTickets()` retrieves all open order tickets. The `pTickets[]` array parameter must be passed by reference. Here's an example of how to retrieve all open buy order tickets:

```
int magicNumber = 123;
ulong buyTickets[];
Positions.GetBuyTickets(magicNumber, buyTickets);
```

After execution of the `GetBuyTickets()` function, the `buyTickets[]` array will contain all of the open buy order tickets that match the magic number in the `magicNumber` variable. To iterate through your order tickets, use a `for` loop with the `ArraySize()` function:

```
for(int i = 0; i < ArraySize(buyTickets); i++)
{
      PositionSelectByTicket(buyTickets[i]);
```

```
        // Perform further order processing
    }
```

The example above uses `ArraySize()` to return the number of order tickets in the `buyTickets` array. The `PositionSelectByTicket()` function then selects each order ticket in `buyTickets[]` using the array index.

Position Information Functions

The position information functions in `Trade.mqh` (`PositionType()`, `PositionStopLoss()`, etc.) have been overridden in the `TradeHedge.mqh` file to accept a ticket number. To use these functions on hedging accounts, pass a ticket number instead of a symbol:

```
ENUM_POSITION_TYPE orderType = PositionType(ticket);
```

The example above does the same thing as the code example under `PositionSelectByTicket()` above. We assume that the `ticket` variable holds a valid order ticket number.

The CTrailing Class

The trailing stop and break even stop class described in Chapter 14 has been updated to support hedging orders. Similar to the trade functions above, the `TrailingStop()` and `BreakEvenStop()` functions take an order ticket as the first parameter:

```
bool TrailingStop(ulong pTicket, int pTrailPoints, int pMinProfit = 0,
      int pStep = 10);
bool TrailingStop(ulong pTicket, double pTrailPrice, int pMinProfit = 0,
      int pStep = 10);
bool BreakEven(ulong pTicket, int pBreakEven, int pLockProfit=0);
```

Use the `CPositions::GetTickets()` function described above to get all open order tickets and iterate through the returned array to adjust the trailing stop for each of your open orders.

Further Examples

The `Experts\Bands RSI CounterTrend (Hedging).mq5` file demonstrates much of the above functionality and how it works in a functioning expert advisor program.