In this appendix you will find the description of the 25 MQL4 trading functions. I decided to write this appendix before writing the third part of “Your First Expert Advisor” lesson because you have to know these important functions before cracking the remaining of the code.

**OrderSend:**

**Syntax:**

```c
int OrderSend( string symbol, int cmd, double volume, double price, int slippage, double stoploss, double takeprofit, string comment=NULL, int magic=0, datetime expiration=0, color arrow_color=CLR_NONE)
```

**Description:**

The `OrderSend` function used to open a sell/buy order or to set a pending order. It returns the ticket number of the order if succeeded and -1 in failure. Use `GetLastError` function to get more details about the error.

**Note:** The ticket number is a unique number returned by `OrderSend` function which you can use later as a reference of the opened or pending order (for example you can use the ticket number with `OrderClose` function to close that specific order).

**Note:** `GetLastError` function returns a predefined number of the last error occurred after an operation (for example when you call `GetLastError` after `OrderSend` operation you will get the error number occurred while executing `OrderSend`).

Calling `GetLastError` will reset the last error number to 0.

You can find a full list of MQL4 errors numbers in `stderror.mqh` file. And you can get the error description for a specific error number by using `ErrorDescription` function which defined at `stdlib.mqh` file.
**Parameters:**

This function takes 11 parameters:

- **string symbol:**
The symbol name of the currency pair you trading (Ex: EURUSD and USDJPY).

**Note:** Use `Symbol()` function to get currently used symbol and `OrderSymbol` function to get the symbol of current selected order.

- **int cmd:**
An integer number indicates the type of the operation you want to take; it can be one of these values:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP_BUY</td>
<td>0</td>
<td>Buying position.</td>
</tr>
<tr>
<td>OP_SELL</td>
<td>1</td>
<td>Selling position.</td>
</tr>
<tr>
<td>OP_BUYLIMIT</td>
<td>2</td>
<td>Buy limit pending position.</td>
</tr>
<tr>
<td>OP_SELLLIMIT</td>
<td>3</td>
<td>Sell limit pending position.</td>
</tr>
<tr>
<td>OP_BUYSTOP</td>
<td>4</td>
<td>Buy stop pending position.</td>
</tr>
<tr>
<td>OP_SELLSTOP</td>
<td>5</td>
<td>Sell stop pending position.</td>
</tr>
</tbody>
</table>

**Note:** You can use the integer representation of the value or the constant name. For example: `OrderSend(Symbol(),0,...)` is equal to `OrderSend(Symbol(),OP_BUY,...)`.
But it's recommended to use the constant name to make your code clearer.

- **double volume:**
The number of lots you want to trade.

- **double price:**
The price you want to open the order at.
Use the functions `Bid` and `Ask` to get the current bid or ask price.

- **int slippage:**
The slippage value you assign to the order.

**Note:** `slippage` is the difference between estimated transaction costs and the amount actually paid.
`slippage` is usually attributed to a change in the spread. ([Investopedia.com](https://www.investopedia.com)).

- **double stoploss:**
The price you want to close the order at in the case of losing.

- **double takeprofit:**
The price you want to close the order at in the case of making profit.
string comment:
The comment string you want to assign to your order (Figure 1).
The default value is NULL which means there's no comment assigned to the order.

**Note:** Default value of a parameter means you can leave (don’t write) it out, and
MQL4 will use a predefined value for this parameter.
For example we can write OrderSend function with or without comment parameter
like this:
OrderSend(Symbol(), OP_BUY, Lots, Ask, 3, Ask-25*Point, Ask+25*Point, "My order
comment", 12345, 0, Green);
Or like this:
OrderSend(Symbol(), OP_BUY, Lots, Ask, 3, Ask-
25*Point, Ask+25*Point, 12345, 0, Green);

int magic:
The magic number you assign to the order.

**Note:** Magic number is a number you assign to your order(s) as a reference enables
you to distinguish between the different orders. For example the orders you have
opened by your expert advisor and the orders have opened manually by the user.

datetime expiration:
The time you want your pending order to expire at.
The default time is 0 which means there's no expiration.

**Note:** The time here is the server time not your local time, to get the current server
time use CurTime function and to get the local time use LocalTime function.

color arrow_color:
The color of opening arrow (Figure 2), the default value is CLR_NONE which means
there's no arrow will be drawn on the chart.
**Example:**

```c
int ticket;
if(iRSI(NULL,0,14,PRICE_CLOSE,0)<25)
{
    ticket=OrderSend(Symbol(),OP_BUY,1,Ask,3,Ask-25*Point,Ask+25*Point,"My order #2",16384,0,Green);
    if(ticket<0)
    {
        Print("OrderSend failed with error ",GetLastError());
        return(0);
    }
}
```

---

**OrderModify:**

**Syntax:**

```c
bool OrderModify( int ticket, double price, double stoploss, double takeprofit, 
datetime expiration, color arrow_color=CLR_NONE)
```
**Description:**

The `OrderModify` function used to modify the properties of a specific opened order or pending order with the new values you pass to the function. It returns true if the order successfully modified and false in failure. Use `GetLastError` function to get more details about the error.

**Parameters:**

This function takes 6 parameters:

- **int ticket:**
  The ticket number of the order you want to modify.

  Note: This number has been assigned to the order by the function `OrderSend`. You can use the function `OrderTicket` to get the ticket number of the current order.

- **double price:**
  The price you want to set for the order.

  Note: Use the function `OrderOpenPrice` to get the open price for the current order.

- **double stoploss:**
  The price you want to close the order at in the case of losing.

- **double takeprofit:**
  The price you want to close the order at in the case of making profit.

  Note: We usually use the `OrderModify` function to change the `stoploss` and/or `takeprofit` values and that called `Trailing`.

- **datetime expiration:**
  The time you want your pending order to expire at.
  Use 0 if you don't want to set an expiration time.

- **color arrow_color:**
  The color of the arrow, the default value is `CLR_NONE` which means there's no arrow will be drawn on the chart.

**Example:**

```
if(TrailingStop>0)
{
    SelectOrder(12345,SELECT_BY_TICKET);
    if(Bid-OrderOpenPrice()>Point*TrailingStop)
    {
        if(OrderStopLoss()<Bid-Point*TrailingStop)
        {
```
OrderClose:

Syntax:

```c
bool OrderClose( int ticket, double lots, double price, int slippage, color Color=CLR_NONE)
```

Description:

The `OrderClose` function used to close a specific opened order (by its ticket). It returns true if the order successfully closed and false in failure. Use `GetLastError` function to get more details about the error.

Parameters:

This function takes 5 parameters:

- **int ticket**: The ticket number of the order you want to close.
- **double lots**: The number of lots you use in the order.
- **double price**: The price you want to open the order at. Use the functions `Bid` and `Ask` to get the current bid or ask price.
- **int slippage**: The `slippage` value of the order.
- **color Color**: The color of closing arrow, the default value is `CLR_NONE` which means there's no arrow will be drawn on the chart.

Note: Use the `OrderLots` function to get the lots value of the current order.
**Example:**

```c
if(iRSI(NULL,0,14,PRICE_CLOSE,0)>75)
{
   OrderClose(order_id,1,Ask,3,Red);
   return(0);
}
```

**OrderSelect:**

**Syntax:**

```c
tool OrderSelect( int index, int select, int pool=MODE_TRADES)
```

**Description:**

The `OrderSelect` function used to select an opened order or a pending order by the ticket number or by index.
It returns true if the order successfully selected and false in failure.
Use `GetLastError` function to get more details about the error.

**Note:** You have to use `OrderSelect` function before the trading functions which takes no parameters:
`OrderMagicNumber, OrderClosePrice, OrderCloseTime, OrderOpenPrice, OrderOpenTime, OrderComment, OrderCommission, OrderExpiration, OrderLots, OrderPrint, OrderProfit, OrderStopLoss, OrderSwap, OrderSymbol, OrderTakeProfit, OrderTicket and OrderType`

**Parameters:**

This function takes 3 parameters:

- **int index:**
The index or the ticket number of the order you want to select. It depends on the second parameter (selecting type).

- **int select:**
The type of selecting operation (by index or by ticket number).
It can be one of two values:
- `SELECT_BY_POS` use the position (index) of the order.
- `SELECT_BY_TICKET` – use the ticket number of the order.

- **int pool:**
If you used the `SELECT_BY_POS` selecting type, you have to determine which pool (data) you will select from:
MODE_TRADES: select from the currently trading orders (opened and pending orders). This is the default value.

MODE_HISTORY: select from the history (closed and canceled orders).

Example:

```cpp
if(OrderSelect(12470, SELECT_BY_TICKET)==true)
    {
    Print("order #12470 open price is ", OrderOpenPrice());
    Print("order #12470 close price is ", OrderClosePrice());
    }
else
    Print("OrderSelect failed error code is", GetLastError());
```

OrderDelete:

Syntax:

```cpp
bool OrderDelete( int ticket)
```

Description:

The OrderDelete function used to delete a pending order. It returns true if the order successfully deleted and false in failure. Use GetLastError function to get more details about the error.

Parameters:

This function takes only 1 parameter:

int ticket:
The ticket number of the order you want to delete.

Example:

```cpp
if(Ask>var1)
    {
    OrderDelete(order_ticket);
    return(0);
    }
```
**OrderCloseBy:**

**Syntax:**

```c
bool OrderCloseBy(int ticket, int opposite, color Color=CLR_NONE)
```

**Description:**

The `OrderCloseBy` function used to close a specific opened order by opening an opposite direction order. It returns true if the order successfully closed and false in failure. Use `GetLastError` function to get more details about the error.

**Parameters:**

This function takes 3 parameters:

- `int ticket`:
  The ticket number of the order you want to close.

- `int opposite`:
  The ticket number of the order you want to open in the opposite direction.

- `color Color`:
  The color of closing arrow, the default value is CLR_NONE which means no arrow will be drawn on the chart.

**Example:**

```c
if(iRSI(NULL,0,14,PRICE_CLOSE,0)>75)
{
    OrderCloseBy(order_id,opposite_id);
    return(0);
}
```

**OrderType:**

**Syntax:**

```c
int OrderType()
```

**Description:**

The `OrderType` function returns the type of selected order that will be one of: `OP_BUY, OP_SELL, OP_BUYLIMIT, OP_BUYSTOP, OP_SELLLIMIT` or `OP_SELLSTOP` (see `OrderSend` function)
The order must be selected by *OrderSelect* before calling *OrderType*.

**Parameters:**

This function doesn’t take any parameters and returns an integer date type (the type of selected order).

**Example:**

```c
int order_type;
if(OrderSelect(12, SELECT_BY_POS)==true)
{
    order_type=OrderType();
    // ...
}
else
    Print("OrderSelect() returned error - ",GetLastError());
```

**HistoryTotal:**

**Syntax:**

```c
int HistoryTotal() 
```

**Description:**

The *HistoryTotal* function searches the account history loaded in the terminal and returns the number of closed orders.

**Note:** We usually use this function with the *OrderSelect* function to get information about a specific order in the history.

**Parameters:**

This function doesn’t take any parameters and returns an integer (the number of closed orders in the history). Use *GetLastError* function to get more details about the error.

**Example:**

```c
// retrieving info from trade history
int i, hstTotal=HistoryTotal();
for(i=0;i<hstTotal;i++)
{
    //---- check selection result
```
if(OrderSelect(i,SELECT_BY_POS,MODE_HISTORY)==false)
    {
        Print("Access to history failed with error (",GetLastError(),")");
        break;
    }
    // some work with order

---

**OrderClosePrice:**

**Syntax:**

```plaintext
double OrderClosePrice()
```

**Description:**

The `OrderClosePrice` function returns the close price of selected order. The order must be selected by `OrderSelect` before calling `OrderClosePrice`.

**Parameters:**

This function doesn’t take any parameters and returns a double data type (the close price of the selected order).

**Example:**

```plaintext
if(OrderSelect(ticket,SELECT_BY_POS)==true)
    Print("Close price for the order ",ticket," = ",OrderClosePrice());
else
    Print("OrderSelect failed error code is",GetLastError());
```

---

**OrderCloseTime:**

**Syntax:**

```plaintext
datetime OrderCloseTime()
```

**Description:**

The `OrderCloseTime` function returns the close time of the selected order. If the return value is 0 that means the order hasn’t been closed yet otherwise it has been closed and retrieved from the history. The order must be selected by `OrderSelect` before calling `OrderCloseTime`. 
**Parameters:**

This function doesn’t take any parameters and returns a datetime data type (the close time of the selected order).

**Example:**

```plaintext
if(OrderSelect(10,SELECT_BY_POS,MODE_HISTORY)==true)
    {
        datetime ctm=OrderOpenTime();
        if(ctm>0) Print("Open time for the order 10 ", ctm);
        ctm=OrderCloseTime();
        if(ctm>0) Print("Close time for the order 10 ", ctm);
    }
else
    Print("OrderSelect failed error code is",GetLastError());
```

---

**OrderComment:**

**Syntax:**

```plaintext
string OrderComment() 
```

**Description:**

The `OrderCloseTime` function returns the comment string for the selected order.

**Note:** This comment has been assigned when you opened the order with `OrderSend` or has been assigned by the server. Sometimes the server append its comment at the end of you comment string.

The order must be selected by `OrderSelect` before calling `OrderCloseTime`.

**Parameters:**

This function doesn’t take any parameters and returns a string data type (the comment string of the selected order).

**Example:**

```plaintext
string comment; 
if(OrderSelect(10,SELECT_BY_TICKET)==false)
    {
        Print("OrderSelect failed error code is",GetLastError());
    }
```
OrderCommission:

Syntax:

```c
double OrderCommission()
```

Description:

The `OrderCommission` function returns the commission amount of the selected order. The order must be selected by `OrderSelect` before calling `OrderCommission`.

Parameters:

This function doesn’t take any parameters and returns a double data type (the commission amount of the selected order).

Example:

```c
if(OrderSelect(10,SELECT_BY_POS)==true)
    Print("Commission for the order 10 ",OrderCommission());
else
    Print("OrderSelect failed error code is",GetLastError());
```

OrderExpiration:

Syntax:

```c
datetime OrderExpiration()
```

Description:

The `OrderExpiration` function returns the expiration time of the selected pending order that you have set in `OrderSend`. The order must be selected by `OrderSelect` before calling `OrderExpiration`. 
**Parameters:**

This function doesn’t take any parameters and returns a datetime data type (the expiration time of the selected pending order).

**Example:**

```cpp
if(OrderSelect(10, SELECT_BY_TICKET)==true)
    Print("Order expiration for the order #10 is ",OrderExpiration());
else
    Print("OrderSelect failed error code is",GetLastError());
```

**OrderLots:**

**Syntax:**

```cpp
double OrderLots()
```

**Description:**

The `OrderLots` function returns the lots value of the selected order that you have set in `OrderSend (volume parameter)`. The order must be selected by `OrderSelect` before calling `OrderLots`.

**Parameters:**

This function doesn’t take any parameters and returns a datetime data type (the lots value of the selected order).

**Example:**

```cpp
if(OrderSelect(10,SELECT_BY_POS)==true)
    Print("lots for the order 10 ",OrderLots());
else
    Print("OrderSelect failed error code is",GetLastError());
```

**OrderMagicNumber:**

**Syntax:**

```cpp
int OrderMagicNumber()
```
**Description:**

The `OrderMagicNumber` function returns the magic number of the selected order that you have set in `OrderSend`. The order must be selected by `OrderSelect` before calling `OrderMagicNumber`.

**Parameters:**

This function doesn’t take any parameters and returns an integer data type (the magic number of the selected order).

**Example:**

```plaintext
if(OrderSelect(10, SELECT_BY_POS)==true)  
    Print("Magic number for the order 10 ", OrderMagicNumber());
else
    Print("OrderSelect failed error code is", GetLastError());
```

---

**OrderOpenPrice:**

**Syntax:**

```plaintext
double OrderOpenPrice( )
```

**Description:**

The `OrderOpenPrice` function returns the open price of the selected order. The order must be selected by `OrderSelect` before calling `OrderOpenPrice`.

**Parameters:**

This function doesn’t take any parameters and returns a double data type (the open price of the selected order).

**Example:**

```plaintext
if(OrderSelect(10, SELECT_BY_POS)==true)  
    Print("open price for the order 10 ", OrderOpenPrice());
else
    Print("OrderSelect failed error code is", GetLastError());
```
**OrderOpenTime:**

**Syntax:**

```plaintext
datetime OrderOpenTime( )
```

**Description:**

The `OrderOpenTime` function returns the open time of the selected order. The order must be selected by `OrderSelect` before calling `OrderOpenTime`.

**Parameters:**

This function doesn’t take any parameters and returns a datetime data type (the open time of the selected order).

**Example:**

```plaintext
if(OrderSelect(10, SELECT_BY_POS)==true)
    Print("open time for the order 10 ",OrderOpenTime());
else
    Print("OrderSelect failed error code is",GetLastError());
```

---

**OrderPrint:**

**Syntax:**

```plaintext
void OrderPrint( )
```

**Description:**

The `OrderPrint` function prints the selected order data to the expert log file. The order must be selected by `OrderSelect` before calling `OrderPrint`.

**Parameters:**

This function doesn’t take any parameters and doesn’t return any value (void).

**Note:** `void` means the function doesn’t return any value, so, you can’t assign it to a variable like this:

```plaintext
int i = OrderPrint(); //no meaning line, although the compiler will not complain.
```
**Example:**

```plaintext
defaultOrderProfit() =
  if(OrderSelect(10, SELECT_BY_TICKET)==true)
      OrderPrint();
  else
      Print("OrderSelect failed error code is",GetLastError());
```

---

**OrderProfit:**

**Syntax:**

```plaintext
double OrderProfit() =
```

**Description:**

The `OrderProfit` function returns the profit of the selected order. The order must be selected by `OrderSelect` before calling `OrderProfit`.

**Parameters:**

This function doesn’t take any parameters and returns a double data type (the profit of the selected order).

**Example:**

```plaintext
defaultOrderStopLoss() =
  if(OrderSelect(10, SELECT_BY_POS)==true)
      Print("Profit for the order 10 ",OrderProfit());
  else
      Print("OrderSelect failed error code is",GetLastError());
```

---

**OrderStopLoss:**

**Syntax:**

```plaintext
double OrderStopLoss() =
```

**Description:**

The `OrderStopLoss` function returns the `stoploss` price of the selected order that you have set in `OrderSend` or modified with `OrderModify`. The order must be selected by `OrderSelect` before calling `OrderStopLoss`. 
**Parameters:**

This function doesn’t take any parameters and returns a double data type (the *stoploss* price of the selected order).

**Example:**

```c
if(OrderSelect(ticket,SELECT_BY_POS)==true)
    Print("Stop loss value for the order 10 ", OrderStopLoss());
else
    Print("OrderSelect failed error code is", GetLastError());
```

**OrdersTotal:**

**Syntax:**

```c
int OrdersTotal()
```

**Description:**

The *OrdersTotal* function returns the number of opened and pending orders. If this number is **0** that means there are no orders (market or pending ones) has been opened.

**Parameters:**

This function doesn’t take any parameters and returns an integer data type (the number of opened and pending orders).

**Example:**

```c
int handle=FileOpen("OrdersReport.csv",FILE_WRITE|FILE_CSV,"\t");
if(handle<0) return(0);
    // write header
    FileWrite(handle,"#","open price","open time","symbol","lots");
int total=OrdersTotal();
    // write open orders
    for(int pos=0;pos<total;pos++)
        { if(OrderSelect(pos,SELECT_BY_POS)==false) continue;

        FileWrite(handle,OrderTicket(),OrderOpenPrice(),OrderOpenTime(),OrderSymbol(),OrderLots());
        }
    FileClose(handle);
```
**OrderSwap:**

**Syntax:**

double OrderSwap( )

**Description:**

The `OrderSwap` function returns the swap value of the selected order. The order must be selected by `OrderSelect` before calling `OrderSwap`.

A *swap* involves the exchange of principal and interest in one currency for the same in another currency. Currency swaps were originally done to get around the problem of exchange controls. ([Investopedia.com](https://www.investopedia.com)).

**Parameters:**

This function doesn’t take any parameters and returns a double data type (the swap value of the selected order).

**Example:**

```cpp
if(OrderSelect(order_id, SELECT_BY_TICKET)==true)
    Print("Swap for the order ", order_id, ", ",OrderSwap());
else
    Print("OrderSelect failed error code is", GetLastError());
```

---

**OrderSymbol:**

**Syntax:**

`string OrderSymbol( )`

**Description:**

The `OrderSymbol` function returns the string representation of currency pair of the selected order (Ex: EURUSD and USDJPY). The order must be selected by `OrderSelect` before calling `OrderSymbol`.

**Parameters:**

This function doesn’t take any parameters and returns a string data type (the string representation of currency pair of the selected order).
Example:

```c
if(OrderSelect(12, SELECT_BY_POS)==true)
    Print("symbol of order ", OrderTicket(), " is ", OrderSymbol());
else
    Print("OrderSelect failed error code is",GetLastError());
```

**OrderTakeProfit:**

**Syntax:**

```c
double OrderTakeProfit()```

**Description:**

The `OrderTakeProfit` function returns the takeprofit price of the selected order that you have set in `OrderSend` or modified with `OrderModify`. The order must be selected by `OrderSelect` before calling `OrderTakeProfit`.

**Parameters:**

This function doesn’t take any parameters and returns a double data type (the takeprofit price of the selected order).

**Example:**

```c
if(OrderSelect(12, SELECT_BY_POS)==true)
    Print("Order ",OrderTicket()," profit: ", OrderTakeProfit());
else
    Print("OrderSelect() failed - ",GetLastError());
```

**OrderTicket:**

**Syntax:**

```c
int OrderTicket()```

**Description:**

The `OrderTicket` function returns the ticket number of the selected order. The order must be selected by `OrderSelect` before calling `OrderTicket`. 
**Parameters:**

This function doesn’t take any parameters and returns an integer data type (the ticket number of the selected order).

**Example:**

```plaintext
if(OrderSelect(12, SELECT_BY_POS)==true)
    order=OrderTicket();
else
    Print("OrderSelect failed error code is",GetLastError());
```

I hope the trading functions are clearer now.

I welcome very much your questions and suggestions.

**Coders’ Guru**

20-12-2005