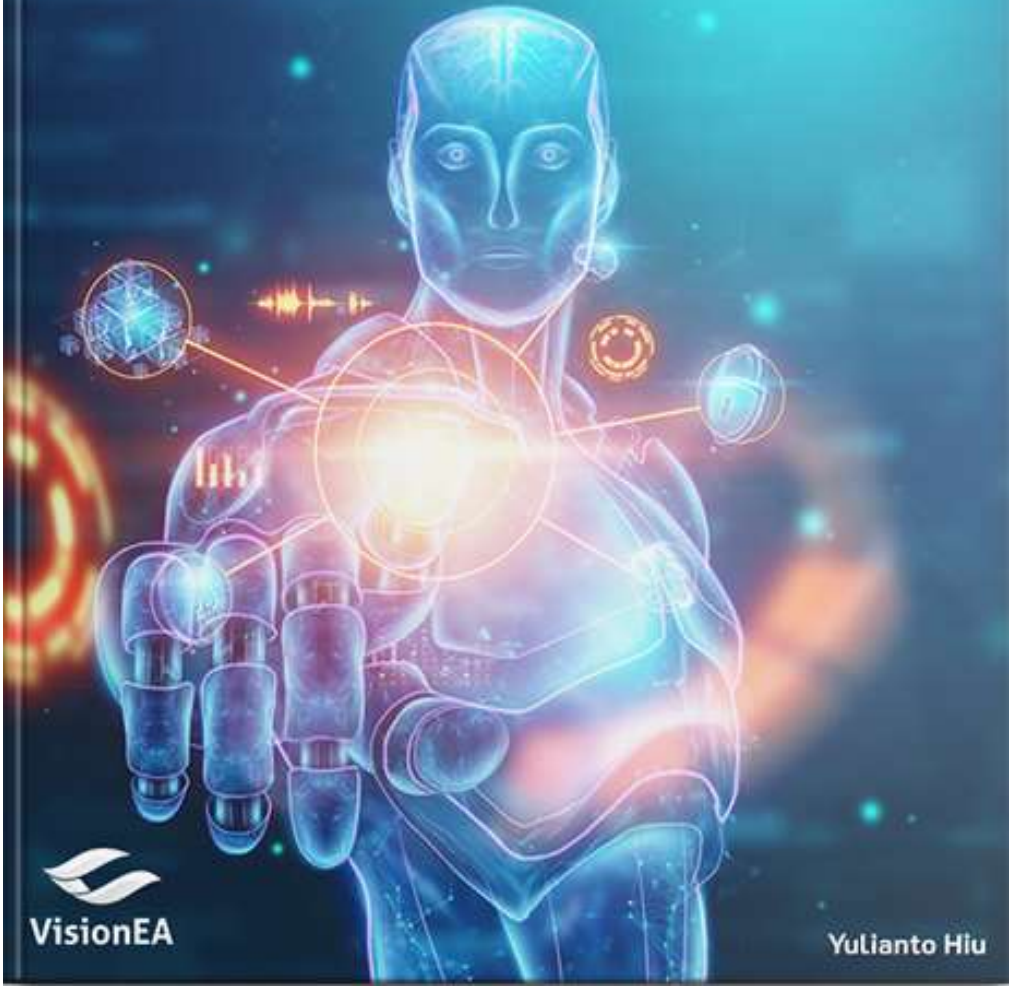
PEMROGRAMAN

Expert Advisor

Mahir membuat sistem perdagangan otomatis pada platform MetaTrader 4 dengan MQL

PRO EDITION



VisionEA

Yulianto Hiu

Pemrograman

Expert Advisor

Mahir membuat sistem perdagangan otomatis pada platform MetaTrader 4 dengan MQL

Pro Edition

Yulianto Hiu

DAFTAR ISI

Menghitung Level Verifikasi

Menambahkan StopLoss dan TakeProfit

Membuka Posisi Lanjutan

Kompatible dengan ECN/STP

Sebagai pemahaman awal, pembagian broker menurut eksekusinya dibagi menjadi 3, yaitu broker Dealing Desk, Non Dealing Desk, dan Hybrid. Broker Dealing Desk kurang diminati karena dianggap sebagai broker bandar. Broker Non Dealing Desk akan melempar order langsung ke pasar dengan menggunakan harga yang lebih real dan meminimalisir manipulasi dari broker. Broker non Dealing Desk ini mampu menawarkan fasilitas ECN (Electronic Communication Network) atau STP (Straight Through Processing), salah satu dari itu. Kriterial membuka akun ECN cukup tinggi seperti membutuhkan minimal deposit yang cukup besar dan membebankan biaya trading melalui komisi bukan spread. Sedangkan STP lebih memberikan kriteria dengan minimal deposit yang relatif kecil dengan membebankan biaya trading dengan nilai spread yang lebih besar. Dan Broker Hybrid adalah broker yang menawarkan perpaduan ECN dan STP.

Dari sisi eksekusi penempatan order pada beberapa jenis baru dari tipe akun ECN yang sering kali tidak bisa melakukan penempatan harga stoploss dan takeprofit bersamaan dengan menggunakan fungsi **OrderSend()**. Padahal kita membutuhkan penempatan harga stoploss dan takeprofit pada pesanan yang kita buat. Karena itu kita membutuhkan fungsi **OrderModify()**.

Merubah Pesanan

Setelah sebuah pesanan terpasang, kita masih bisa mengubah harga stoploss, takeprofit, merubah pending order dan expiration time dengan fungsi **OrderModify()**.

Untuk merubah sebuah pesanan, kita membutuh nomor tiket pesanan tersebut. Adapun syntax **OrderModify()** adalah:

```
bool  OrderModify(
  int       ticket,     // ticket
  double    price,      // price
  double    stoploss,   // stop loss
  double    takeprofit, // take profit
  datetime  expiration, // expiration
  color     arrow_color // color
  );
```

- Ticket : nomor unik dari sebuah pesanan
- Price : harga baru untuk pending order
- Stoploss : harga baru untuk stoploss
- Takeprofit : harga baru untuk takeprofit
- Expiration : waktu kadaluarwarsa pending order, tidak semua broker mendukung parameter ini.
- Color : warna panah untuk menandakan level harga stoploss/takeprofit.

Jika fungsi ini berhasil, akan mengembalikan nilai **true** sebaliknya nilai **false**. Dan jika terjadi kesalahan, informasi detil dari kesalahan bisa diketahui dengan memanggil fungsi **GetLastError()**.

Untuk pesanan yang sudah didalam pasar, kita bisa menggunakan fungsi OrderModify() untuk mengubah harga stoploss dan harga takeprofit sedangkan untuk pesanan pending kita masih dapat merubah harga pesanan, stoploss dan takeprofit.

Kita akan pelajari bagaimana penerapan perubahan harga stoploss dan takeprofit pada posisi pesanan order yang sudah ada.

Menambahkan StopLoss dan TakeProfit terhadap Pesanan yang telah ada

Sebelum menambahkan StopLoss maupun TakeProfit pastikan bahwa pesanan yang anda lakukan telah berhasil dan benar. Hal ini dilakukan untuk memastikan bahwa nilai pengembalian fungsi **OrderSend** (), yang merupakan nomor tiket terhadap pesanan yang dilakukan. Apabila pesanan yang dilakukan tidak berhasil dan terjadi eror, maka nomor tiket akan sama dengan **-1**.

Fungsi lain yang kami gunakan yakni, **OrderSelect**() yang berfungsi untuk mengambil informasi pesanan yang baru saja dilakukan. Fungsi **OrderOpenPrice**(), **OrderTakeProfit**(), **OrderStopLoss**() dan **OrderExpiration**() akan digunakan apabila terdapat nilai yang tidak berubah pada fungsi **OrderModify**() dan **OrderModify**() akan digunakan untuk menambahkan StopLoss dan TakeProfit terhadap pesanan.

Berikut contoh cara menambahkan StopLoss dan TakeProfit menggunakan fungsi **OrderModify**() yang dihitung setelah memindahkan perhitungan StopLoss dan TakeProfit dengan fungsi **OrderSend**() :

```
int BuyTicket = OrderSend(Symbol(),OP_BUY,LotSize,Ask,UseSlippage,0,0,
"Buy Order",MagicNumber,0,Green);
if(BuyTicket > 0)
{
OrderSelect(BuyTicket,SELECT_BY_TICKET);
double OpenPrice = OrderOpenPrice();
if(StopLoss > 0) double BuyStopLoss = OpenPrice – (StopLoss * UsePoint);
if(TakeProfit > 0) double BuyTakeProfit = OpenPrice + (TakeProfit *
UsePoint);
if(BuyStopLoss > 0 || BuyTakeProfit > 0)
```

```
        {
        bool TicketMod = OrderModify( BuyTicket, OrderOpenPrice(),
BuyStopLoss, BuyTakeProfit,0);
        }
}
```

Fungsi **OrderSend**() identik dengan contoh sebelumnya, dan terkecuali apabila kita menggunakan nilai 0 untuk parameter StopLoss dan TakeProfit. Nilai 0 (nol) menandakan bahwa tidak ada StopLoss maupun TakeProfit yang ditambahkan pada pesanan. **BuyTicket** berfungsi menyimpan nomor tiket pada pesanan.

Fungsi **if** berfungsi dalam memeriksa **BuyTicket** sudah benar dan valid, dalam artian lebih besar dari 0, kemudian menggunakan fungsi **OrderSelect**() dengan nomor **BuyTicket** dan mengambil harga pembukaan menggunakan **OrderOpenPrice**() kemudian menetapkannya ke variabel **OpenPrice**.

Kemudian dalam perhitungan StopLoss dan TakeProfit masih berhubungan dengan harga pembukaan pesanan yang kita lakukan, kemudian akan dilakukan pemeriksaan terhadap variabel eksternal **StopLoss** dan **TakeProfit** telah lebih besar dari pada 0 (nol). Jika begitu, barulah kemudian harga StopLoss dan TakeProfit dapat dihitung.

Fungsi **OrderModify**() berfungsi untuk menambahkan StopLoss dan TakeProfit terhadap pesanan. Sebelumnya akan dipastikan bahwa variabel **BuyStopLoss** dan **BuyTakeProfit** bukan atau lebih dari nol. Apabila diharuskan untuk memodifikasi pesanan dengan nilai tidak berubah, maka akan muncul kode eror 1 dari fungsi **OrderModify**().

Nomor pada **BuyTicket** merupakan parameter pertama pada **OrderModify**(), selain itu kita juga dapat menggunakan **OrderTicket**() dan untuk parameter kedua adalah harga baru pesanan. Karena kita tidak melakukan modifikasi terhadap harga

pesanan, maka digunakan fungsi **OrderOpenPrice**() untuk menunjukkan bahwa harga pesanan tidak berubah.

Harus diingat bahwa kita hanya dapat memodifikasi harga pesanan pada pesanan yang masih tertunda, dan dapat melampaui nilai pada parameter **Price**, karena harga pasar pada pesanan tidak dapat diganti. Akan tetapi, tidak dapat diasumsikan bahwa pesanan pasar akan dirubah terus menerus, jadi kami menggunakan **OpenOrderPrice**().

Variable **BuyStopLoss** dan **BuyTakeProfit** meneruskan nilai StopLoss dan TakeProfit yang berubah dengan menggunakan fungsi **OrderModify**(). Jika anda berencana untuk menggunakan waktu kadarluwasa pesanan untuk pesanan yang tertunda, anda dapat menggunakan **OrderExpiration**() sebagai parameter kadarluwasa yang tidak berubah.

Meskipun metode ini menambahkan beberapa langkah, namun langkah ini direkomendasikan karena metode ini menempatkan StopLoss dan TakeProfit pada pesanan pada expert advisor untuk memastikan bahwa itu kompatibel dengan broker lainnya. Metode ini juga memiliki keunggulan yang memungkinkan kami untuk letakkan stop loss yang akurat dan ambil harga untung tanpa efek slippage

Memodifikasi Harga Pesanan Tertunda

OrderModify() dapat digunakan untuk memodifikasi harga pesanan yang tertunda. Apabila pesanan tertunda telah tercapai dan pesanan telah terisi, maka tak dapat lagi dikatakan sebagai pesanan tertunda dan harganya pun tidak dapat diganti lagi.

Kami menggunakan variabel **NewPendingPrice** yang mewakili penggantian harga pesanan. Kami mengasumsikan bahwa harganya telah dihitung dan valid, berikut cara memodifikasi harga pesanan yang tertunda.

OrderSelect(Ticket,SELECT_BY_TICKET);

if(NewPendingPrice != OrderOpenPrice())

{

*bool TicketMod = OrderModify (**Ticket,NewPendingPrice,OrderStopLoss(),***

***OrderTakeProfit**(),0);*

}

Cara kami mengembalikan informasi yakni menggunakan **OrderSelect**(). Dengan cara ini kami dapat melewati StopLoss dan TakeProfit yang tidak berubah pada fungsi **OrderModify**() dan sebelum memodifikasi pesanan, kami akan memeriksa dan memastikan bahwa harga pesanan yang tertunda tidak sama dengan harga pesanan tertunda saat ini.

Pada fungsi **OrderModify**(), kami mespesifikasi tiket pesanan kami, harga pesanan baru akan tersimpan dalam **NewPendingPrice** dan nilai StopLoss dan TakeProfit yang tidak berubah ditampilkan pada **OrderStopLoss**() dan **OrderTakeProfit**(), pada ini tidak ada waktu kadarluwasa jadi kami menggunakan 0 pada parameter kadarluwasa.

Memverikasi Harga StopLoss dan TakeProfit

Dalam sistematika pengambilan minimun TakeProfit pips akan sama dengan harga pembukaan pesanan, plus maupun minus pada stop level. Apabila StopLevelnya adalah 3 dengan harga pembukaan sebesar 1.4500, maka pengambilan harga keuntungan akan berada di atas 1,4503.

Sementara dalam pengambilan minimun StopLoss pips dalam pesanan pasar, akan termasuk dengan spread saat ini, makan pengambilan minimun StopLoss akan lebih besar daripada minimun TakeProfit. Contohnya, apabila StopLevelnya adalah 3

pips, dan Spreadnya adalah 2 pips dengan harga pembukaan sebesar 1.4500, maka StopLoss pada pesanan pasar adalah dibawah 1.4995.

Contoh di atas tidak berlaku pada pesanan yang tertunda, apabila anda ingin memverikasi pesanan tertunda, maka itu tidak terlalu penting untuk memastikan Spread yang ada pada pesanan. Namun, jika anda melakukan pesanan tertunda pada harga 1.4500 dengan StopLevel 3 pips, maka StopLoss dapat ditaruh dengan harga dibawah 1.4497.

Berikut contoh bagaimana cara untuk memeriksa kevalid-an StopLoss dan TakeProfit pada pesanan Buy. Apabila harga StopLoss maupun TakeProfit tidak valid, maka kami akan secara otomatis menyesuaikannya dan menjadi beberapa pips saja di luar dari StopLevel.

*double MinStop = 5 * UsePoint;*

if(BuyStopLoss > LowerStopLevel) BuyStopLoss = LowerStopLevel - MinStop;

if(BuyTakeProfit < UpperStopLevel) BuyTakeProfit = UpperStopLevel + MinStop;

Variabel **MinStop** menambah atau mengurangi 5 pips dari StopLevel, untuk memastikan bahwa harga valid kami tidak berubah karena SlipPage. Anda dapat menyesuaikan nilai ini untuk menerapkan levle stop / profit minimum yang memadai, atau bahkan menggunakan variabel eksternal untuk menyesuaikan jumlah ini.

Pada baris kedua, StopLoss membandingkannya dengan **LowerStopLevel**. Apabila StopLossnya lebih besar daripada LowerStopLevel, maka StopLoss tersebut menjadi tidak valid. Dalam kasus ini, kami menyesuaikan StopLoss menjadi beberapa pips di bawah StopLevel milik kami dan pada baris ketiga, hal yang sama terjadi pada TakeProfit.

Dalam memeriksa StopLoss dan TakeProfit untuk menjual pesanan, perhitungannya secara simpel dibalikkan seperti berikut.

if(SellTakeProfit > LowerStopLevel) SellTakeProfit = LowerStopLevel - MinStop;

if(SellStopLoss < UpperStopLevel) SellStopLoss = UpperStopLevel + MinStop;

Alih-alih secara otomatis menyesuaikan harga yang tidak valid, anda juga dapat menampilkan pesan kesalahan dan menghentikan eksekusi program. Dengan cara ini pengguna akan diminta untuk menyesuaikan pengaturan StopLoss milik mereka atau TakeProfit sebelum melanjutkan. Berikut ini cara untuk melakukannya,

if(BuyStopLoss > LowerStopLevel)

{

Alert("The stop loss setting is too small!");

return(0);

}

Apabila perhitungan StopLoss di atas StopLevel dan mendekati harga, maka fungsi **Alert**() akan muncul sebagai pesan pop-up. Operator **return** akan keluar dari fungsi saat ini dan memastikan bahwa pesanan tidak akan terjadi.

Pada buku ini, harga-harga akan secara otomatis akan menyesuaikan harga yang tidak valid, dengan berasumsi bahwa akan lebih baik apabila menaruh pesanan pada urutan yang benar daripada tidak menaruh satu pun pesanan. Ini akan berguna dalam mencetak pesan log.

if(BuyStopLoss > LowerStopLevel)

{

BuyStopLoss = LowerStopLevel - MinStop;

Print("Stop loss is invalid and has been automatically adjusted");

Memverifikasi Harga Pesanan Tertunda

Pada verifikasi BuyStop ataupun SellLimit pesanan, variabel **PendingPrice** menyimpan harga pesanan yang tertunda. Berikut cara memverifikasinya.

if(PendingPrice < UpperStopLevel) PendingPrice = UpperStopLevel + MinStop;

Menyadari bahwa kode di sini identik dengan kode di atas yang memeriksa harga BuyTakeProfit dan SellStopLoss. Dan inilah kode untuk memeriksa harga pesanan tertunda pada SellStop dan BuyLimit ;

if(PendingPrice > UpperStopLevel) PendingPrice = UpperStopLevel – MinStop;

Menghitung Ukuran Lot

Pada bagian ini akan kita akan mengeskplorasi lebih dalam mengenai metode-metode rumit yang menghitung ukuran lot tergantung pada jumlah maksimum yang anda inginkan. Dalam memilih level StopLoss dan TakeProfit, menggunakan lot yang sesuai adalah salah satu alat manajemen terbaik yang dapat anda miliki. Menentukan ukuran lot dapat sesederhana mendeklarasikan variabel eksternal dan menggunakan ukuran lot tetap untuk setiap pesanan.

Over-Leverage merupakan satu di antara hal yang dapat membuat para trader forex gagal, menggunakan ukuran lot yang besar dapat mengosongkan akun anda dalam seketika dan pula dapat menghasilkan keuntungan yang besar dalam seketika. Direkomendasikan untuk menggunakan tidak lebih dari 2-3% ekuitas anda per transaksi trade yang anda lakukan. Dengan ini maka anda tidak akan kehilangan lebih dari 2-3% per trade pada akun anda.

Manajemen Uang

Dalam metode kali ini, cara menghitung ukuran lot kita membutuhkan persentase ekuitas dan pips pada StopLoss yang spesifik untuk digunakan dan yang digunakan pada metode ini adalah variabel eksternal dari **EquityPercent** untuk mengatur persentase ekuitas. Contoh, anggap bahwa 50 pips yang digunakan, sebagai berikut.

extern double EquityPercent = 2;

extern double StopLoss = 50;

Pertama-tama, kita perlu untuk menghitung jumlah ekuitas yang spesifik dengan **EquityPercent** dan apabila kita memiliki saldo sebesar $10.000 dan akan menggunakan 2% darinya, maka perhitungannya sebagai berikut.

*double RiskAmount = AccountEquity() * (EquityPercent / 100);*

AccountEquity() adalah sebuah fungsi MQL yang mengembalikan ekuitas akun saat ini, kemudian kita membagi **EquityPercent** dengan 100 untuk mendapatkan hasil nilai fraksional (0.02). Setelah itu, kalikan itu dengan **AccountEquity**() untuk menghitung ekuitas yang akan digunakan, 2% dari $10.000 adalah $200, dan ini akan tersimpan di variabel **RiskAmount**.

Langkah selanjutnya, kita harus menemukan TickValue, yang merupakan keuntungan per pips yang kita beli saat trading pada mata uang yang diinginkan. Contoh, apabila kita melakukan trading dengan 1 lot EURUSD pada akun standard (100k lot), maka keuntungannya per pips adalah $10, sementara pada akun mini akan menjadi 10k lots dan profit per pipsnya adalah $1.

Kita dapat menggunakan fungsi **MarketInfo**() dengan parameter **MODE_TICKVALUE** untuk mengembalikan keuntungan per pip pada mata uang yang spesifik. TickValue harus dalam bentuk pips, jadi saat kita melakukan trading broker pip friksional (desimal 3 atau 5), kita harus mengalikannya dengan nilai 10.

double TickValue = MarketInfo(Symbol(),MODE_TICKVALUE);

*if(Point == 0.001 || Point == 0.00001) TickValue *= 10;*

Dengan contoh apabila kita sedang melakukan trading dengan akun standard, maka TickValue untuk EURUSD adalah 10. Ini akan tersimpan pada variabel **TickValue**, akan tetapi itu dalam broker pip friksional maka **TickValue** akan menjadi 1. Kita akan mengalikannya dengan 10 untuk mendapatkan hasil pip yang ekuivalen. Namun apabila variabel **Point** pada mata uang mengindikasikan 3 atau 5 pada desimal maka, **TickValue** akan dikalikan dengan 10 untuk membuatnya sama dengan 2 atau 4 desimal.

Berikut cara untuk menghitung ukuran lot, pertama, melakukan pembagian **RiskAmount** dengan setting **StopLoss.** Ini akan memberikan keuntungan per tick pada pesanan, contohnya $200 dibagi dengan StopLoss sebesar 50 akan memberikan $4. Selanjutnya kita lakukan pembagian **TickValue** untuk mendapatkan ukuran lot.

double CalcLots = (RiskAmount / StopLoss) / TickValue;

Maka, ukuran lot kita yang terhitung saat ini pada akun standard adalah 0,4. Pada akun mini, ukuran lotnya adalah 4. Nilai ini akan tersimpan dalam variabel **CalcLots.**

Apabila anda melakukan pengelolaan uang yang tepat, maka persentase ekuitas anda akan konsisten. (1-2% untuk risiko konservatif hingga 5% untuk risiko yang lebih tinggi). StopLoss anda sangat bergantung kepada TimeFrame dan metode trading yang anda gunakan dan ukuran lot sangat bervariasi tergantung pada StopLoss anda.

StopLoss yang ketat akan menghasilkan ukuran lot yang lebih besar, yang memberikan banyak keuntungan apabila pesanan anda menyentuh TakeProfit. Di lain hal, apabila anda menggunakan StopLoss yang besar, maka ukuran lot nya menjadi lebih kecil dan metode ini akan mendapat keuntungan dari menggunakan StopLoss yang cukup ketat dan / atau nilai TakeProfit yang besar.

Apabila anda menggunakan StopLoss yang besar atau tidak menggunakannya sama sekali, maka ukurannya lot akan pasti dan akan lebih menguntungkan. Kita harus memustukan untuk menghitung ukuran lot atau menggunakan ukuran lot yang pasti. Mari menghitung menggunakan variabel eksternal Boolean, **DynamicLotSize** untuk menghitung ukuran lot.

```
// External variables

extern bool DynamicLotSize = true;

extern double EquityPercent = 2;

extern double FixedLotSize = 0.1;

// Start function

if(DynamicLotSize == true)

{

double RiskAmount = AccountEquity() * (EquityPercent / 100);

double TickValue = MarketInfo(Symbol(),MODE_TICKVALUE);

if(Digits == 3 || Digits == 5) TickValue *= 10;

double CalcLots = (RiskAmount / StopLoss) / TickValue;

double LotSize = CalcLots;

}

else LotSize = FixedLotSize;
```

Apabila **DynamicLotSize**nya telah dipasang dengan benar, maka kita akan menghitung ukuran lotnya berdasarkan StopLoss dan menetapkan nilai tersebut pada variabel **LotSize**. Apabila **DynamicLotSize**nya salah, kita tinggal menetapkan **FixedLotSizenya** pada **LotSize**. Variabel **LotSize** akan ditaruh ke fungsi **OrderSend**() sebagai ukuran lot pada pesanan.

Memverifikasi Ukuran Lot

Seperti pada StopLoss, TakeProfit, dan pesanan yang tertunda, ukuran lot pula harus dapat diverifikasi oleh broker milik anda. Ini menandakan bahwa ukuran lot anda tidak seharusnya terlalu besar atau terlalu kecil dan tidak semestinya terspesifikasi pada lot mikro (0,01) apabila broker anda tidak mendukung tersebut. Anda pula harus menormalkan ukuran lot anda pada desimal yang benar.

Dalam memeriksa minimun dan maksimuk ukuran lot, fungsi **MarketInfo**() menggunakan parameter **MODE_MINLOT** dan **MODE_MAXLOT** untuk dibandingkan dengan ukuran lot saat ini pada ukuran minimun dan maksimum lot. Apabila ukuran lot tidak valid, maka itu akan secara otomatis dirubah ukurannya menjadi minimun atau maksimum.

```
 if(LotSize < MarketInfo(Symbol(),MODE_MINLOT))
{
LotSize = MarketInfo(Symbol(),MODE_MINLOT);
}
else if(LotSize > MarketInfo(Symbol(),MODE_MAXLOT))
{
LotSize = MarketInfo(Symbol(),MODE_MAXLOT);
}
```

Kami membandingkan nilai **LotSize**, lot yang sudah dihitung atau lot yang pasti pada minimun dan maksimum ukuran lot. Apabila **LotSize** kurang dari minimun ukuran lot atau lebih besar daripada nilai maksimum maka itu akan diberikan nilai minimun atau maksimum yang sesuai.

Selanjutnya, kita akan membandingkan ukuran lot milik kita terhadap StepValue, StepValue mengindikasi kan apakah broker mengizinkan lot mikro (0.01)

atau lot mini (0.1). Apabila anda menggunakan lot mikro pada broker yang hanya mengizinkan lot mini, maka akan muncul eror dan trade anda tidak akan berhasil. Berikut kode untuk memeriksa StepValue.

```
if(MarketInfo(Symbol(),MODE_LOTSTEP) == 0.1)
{
LotSize = NormalizeDouble(LotSize,1);
}
else LotSize = NormalizeDouble(LotSize,2);
```

Fungsi **NormalizeDouble**() membulatkan nilai **LotSize** menjadi digit yang ditentukan pada argumen kedua. Pada baris pertama, apabila StepSizenya adalah 0.1 menandakan bahwa brokernya hanya menggunakan lot mini, dan **LotSize** akan dibulatkan menjadi 1 posisi angka desimal dan sebaliknya akan dibulatkan menjadi 2 posisi angka desimal.

Dalam waktu yang akan datang apabila anda bertemu dengan broker yang mengizinkan anda untuk menggunakan ukuran lot pada 3 angka desimal, maka anda akan dengan mudah dapat memodifikasi kode untuk memeriksanya akan tetapi pada umumnya broker MetaTrader hanya menggunakan antara 1 dengan 2 desimal pada ukuran lot.

Konten Trading

MetaTrader memiliki SingleTradeExecutionThread pada ExpertAdvisor yang mengartikan bahwa hanya satu ExpertAdvisor yang dapat melakukan trading dalam satu waktu tak terkecuali seberapa banyak ExpertAdvisor yang sedang berjalan pada

terminal. Sebelum memulai dengan trading kita harus memastikan apakah TradeExecution sedang digunakan.

Fungsi **IsTradeContextBusy**() akan menjadi True apabila TradeExecution sedang digunakan, selain itu maka akan False. Kita akan …….. termasuk **OrderSend**(), **OrderClose**(), **OrderDelete**(), atau **OrderModify**().

Berikut cara untuk memeriksa TradeExecution menggunakan **IsTradeContextBusy**();

while(IsTradeContextBusy()) Sleep(10);

int Ticket = OrderSend(Symbol(),OP_BUY,LotSize,Ask,UseSlippage,0,0,

"Buy Order",MagicNumber,0,Green);

Kami menggunakan loop **while** dalam mengevaluasi **IsTradeContextBusy**(). Apabila fungsinya menjadi benar menandakan bahwa TradeExecution-nya telah digunakan maka ExpertAdvisor akan **Sleep** untuk 10 milidetik. Loop **while** akan terus mengeksekusi **IsTradeContextBusy**() hingga menjadi True. Saat TradeThread nya kosong, trading akan kembali berjalan.

Apabila ExpertAdvisor berusaha untuk melakukan trade saat TradeExecutionThread sedang digunakan maka akan muncul sebuah eror 147 : “TradeContent busy” atau “TradeContent sibuk”. Meskipun cara ini dapat diandalkan dalam menghindari eror “TradeContent busy”, itu tidak mudah, terutama ketika banyak ExpertAdvisor yang berusaha untuk berdagang pada saat yang sama.

Merefresh Variabel yang telah ditentukan

Nilai pada variabel yang telah ditentukan seperti **Bid** dan **Ask** diatur begitu ExpertAdvisor memulai eksekusinya, dan waktu yang dibutuhkan untuk mengeksekusi kode ExpertAdvisor hanya sebentar, terukur dalam milidetik saja. Akan tetapi jika anda menemukan delay pada respon server trading dan pada faktanya harga dapat berubah dengan cepat, oleh karena itu sangat penting bagi anda untuk tetap menggunakan harga saat ini.

Fungsi **RefreshRate**() berfungsi mengupdate konten variabel yang telah ditentukan dengan harga terkini berdasarkan server. Variabel **Bid** dan **Ask** sangat direkomendasikan untuk selalu digunakan, terutama setelah mengeksekusi trade.

Apabila anda mengembalikan harga menggunakan fungsi **MarketInfo**(), itu tidak penting menggunakan **RefreshRate**(). Dalam pembuatan fungsi, kita menggunakan **MarketInfo**() untuk mengembalikan harga ketimbang variabel yang ditetapkan. Namun, anda mungkin akan menggunakan **Ask** dan **Bid** pada fungsi **start** untuk menjadi referensi grafik harga.

Menangani Kesalahan (Error)

Dalam melakukan penggantian, modifikasi, maupun menutup pesanan, eror dapat muncul karena adanya kesalahan pada parameter trading, requotes, atau masalah pada server. Untuk menanganinya kami telah memastikan bahwa parameter trading yang kami gunakan telah valid dan diperiksa secara rutin untuk menghindari eror-eror umum, dan saat muncul eror kami akan mengabari pengguna adanya eror dan cara menanganinya.

Dalam memeriksa eror maupun kesalahan yang terjadi kami menggunakan fungsi output seperti **OrderSend(), OrderModify(),** dan **OrderClose()**. Apabila fungsi-

fungsi tersebut tidak berhasil, maka fungsi ini akan berubah kembali menjadi -1 pada **OrderSend()**, atau **false** pada **OrderModify()** dan **OrderClose().**

Pada sesi ini, mari kita membuat sebuah penangan eror terhadap fungsi **OrderSend().** Apabila nilai **OrderSend()** adalah -1, kami akan melakukan penanganan eror untuk memunculkan pemberitahuan kepada pengguna dan menampilkan parameter trade yang berkaitan dan informasi mengenai harga pada log.

Pertama, kita akan mengembalikan kode eror menggunakan fungsi **GetLastError()** dan mengembalikan nilainya dalam bentuk variabel. Apabila sudah digunakan maka **GetLastError()** erornya akan diperbaiki dan nilainya akan kembali menjadi 0. Kami akan mendeklarasikan variabel global yang disebut **ErrorCode** dan menggunakannya untuk menyimpan nilai **GetLastError().**

Kemudian, untuk mendapatkan informasi deskriptif mengenai eror, data **stdlib.mqh** memiliki fungsi **ErrorDescription(),** fungsi ini mengembalikan sebuah garis dengan penjelasan eror dan kita akan menambahkan pernyataan **#include** pada **stdlib.mqh** yang berfungsi untuk menyimpan nilai **GetLastError()**.

Setelahnya, kami akan menampilkan sebuah pesan peringatan dengan fungsi **Alert()** dan akan tercetak pada log. Pesan tersebut berisi kode eror, penjelasan mengenai eror, dan secara singkat menjelaskan tindakan yang akan dilakukan oleh kami dalam menangani eror. Dengan begitu pula anda akan dapat mengetahui bagian mana yang mengalami eror.

Pada akhirnya, kami akan mencetak informasi harga yang berhubungan pada log menggunakan fungsi **Print()** beriringan dengan harga Bid & Ask begitu pula dengan parameter trade seperti ukuran lot dan harga-harga lainnya.

```
// Preprocessor section

#include <stdlib.mqh>

// Global variable

int ErrorCode;
```

```
// Order placement
int Ticket =
OrderSend(Symbol(),OP_BUYSTOP,LotSize,PendingPrice,UseSlippage,0,0,
"Buy Stop Order",MagicNumber,0,Green);
if(Ticket == -1)
{
ErrorCode = GetLastError();
string ErrDesc = ErrorDescription(ErrorCode);
string ErrAlert = StringConcatenate("Open Buy Stop Order - Error ",
ErrorCode,": ",ErrDesc);
Alert(ErrAlert);

string ErrLog = StringConcatenate("Bid: ",Bid," Ask: ",Ask," Price: ",
PendingPrice," Lots: ",LotSize);
Print(ErrLog);
}
```

Kita akan menambahkan data **stdlib.mqh** pada bagian atas, dan menambahkan variabel **ErrorCode** untuk menyimpan kode eror. **OrderSend()** menaruh BuyStopOrder, dan apabila fungsinya berhasil, maka kode menangani eror akan berjalan.

Langkah pertama, kita akan menggunakan nilai **GetLastError()** dalam **ErrorCode** dan kemudian menggunakan fungsi **ErrorDescription()** dan **ErrorCode**

sebagai argument. Selanjutnya, gunakan fungsi **StringConcatenate()** untuk membuat pesan peringatan yang tersimpan dalam variabel **ErrAlert.**

StringConcatenate() adalah sebuah fungsi MQL yang mengizinkan anda untuk membuat garis dengan variabel dan konstan. Setiap elemen string yang akan digabungkan (atau "digabung") dan dipisahkan oleh koma. Contoh dari pernyataan di atas terdapat pada MetaTrader dapat dilihat pada syntax highlighting.

Anda juga dapat menggabungkan garis dengan tanda plus (+), menggunakan **StringContatenate()** tentu jauh lebih mudah dan efisien, akan tetapi apabila anda ingin menggabungkan garis yang pendek, gunakan tanda plus untuk menggabungkan garis konstan dan variabelnya.

string PlusCat = "The current Ask price is "+Ask;

// Sample output: The current Ask price is 1.4320

Fungsi **TheAlert()** menampilkan pop-up pada desktop pengguna yang berisi konten variabel **ErrAlert,** berikut contoh alertmessage yang akan muncul.

Kami membuat garis lainnya dengan harga dan parameter milik kami kemudian menyimpannya dalam variabel **ErrLog** yang kemudian akan berlanjut ke fungsi **Print()**. **Print()** akan mencetak fungsi argument ke ExpertLog. ExpertLog dapat

anda temukan pada ExpertTab dalam TerminalWindow atau jika anda menggunakan StrategyTester anda dapat menemukannya pada JournalTab dalam TesterWindow.

Berikut isi pada log, baris pertama merupakan output dari fungsi **Alert()**, baris kedua merupakan output dari fungsi **Print().** Menyadari adanya eror, “invalid trade volume” dan pada faktanya ukuran lotnya adalah 0, maka permasalahannya ada pada ukuran lot yang tidak valid.

16:47:54 Profit Buster EURUSD,H1: Alert: Open Buy Stop Order - Error 131:

invalid trade volume

16:47:54 Profit Buster EURUSD,H1: Bid: 1.5046, Ask: 1.5048, Lots: 0

Anda dapat membuat sebuah cara menangani eror dengan fungsi lainnya, terutama dengan fungsi **OrderModify()** dan fungsi **OrderClose()**, anda pula dapat membuat sebuah cara menangani eror yang lebih rumit dengan kode eror.

Contohnya, apabila anda mendapatkan eror dengan kode 130 : “invalid stop”, anda dapat menampilkan pesan “StopLoss dan TakeProfit tidak valid”. Berikut cara untuk melakukannya.

```
ErrorCode = GetLastError();

string ErrDesc;

if(ErrorCode == 129) ErrDesc = "Order opening price is invalid!";

if(ErrorCode == 130) ErrDesc = "Stop loss or take profit is invalid!";

if(ErrorCode == 131) ErrDesc = "Lot size is invalid!";

string ErrAlert = StringConcatenate("Open Buy Order - Error ",ErrorCode,":
",ErrDesc);

Alert(ErrAlert);
```

Proses Menyatukan Semuanya

Pada sesi ini kita akan menyatukan semua fitur yang telah kita bahas di bab-bab sebelumnya menjadi satu dengan sedikit menambahkan beberapa fitur seperti OrderModification, StopLevelVerification, TradeContextChecking, dan variabel yang telah ditetapkan dan ukuran lot pada EA. Berikut cara untuk melakukannya.

```
#property copyright "Yulianto Hiu"

#include <stdlib.mqh>

// External variables

extern bool DynamicLotSize = true;

extern double EquityPercent = 2;

extern double FixedLotSize = 0.1;

extern double StopLoss = 50;

extern double TakeProfit = 100;

extern int Slippage = 5;

extern int MagicNumber = 123;

extern int FastMAPeriod = 10;

extern int SlowMAPeriod = 20;

// Global variables

int BuyTicket;

int SellTicket;

double UsePoint;

int UseSlippage;
```

```
int ErrorCode;
```

Kami menambahkan pernyataan **#include** pada data **stdlib.mqh** yang berisikan fungsi **ErrorDescription()** dalam rutinitas menangani eror. Ada 3 variabel yang kami tambahkan pada pengukuran lot dan variabel global pada kode eror. Berikut kode yang berjalan pada awal fungsi **Start()**.

```
// Moving averages
double FastMA = iMA(NULL,0,FastMAPeriod,0,0,0,0);
double SlowMA = iMA(NULL,0,SlowMAPeriod,0,0,0,0);
// Lot size calculation
if(DynamicLotSize == true)
{
double RiskAmount = AccountEquity() * (EquityPercent / 100);
double TickValue = MarketInfo(Symbol(),MODE_TICKVALUE);
if(Point == 0.001 || Point == 0.00001) TickValue *= 10;
double CalcLots = (RiskAmount / StopLoss) / TickValue;
double LotSize = CalcLots;
}
else LotSize = FixedLotSize;
// Lot size verification
if(LotSize < MarketInfo(Symbol(),MODE_MINLOT))
{
LotSize = MarketInfo(Symbol(),MODE_MINLOT);
```

```
}
else if(LotSize > MarketInfo(Symbol(),MODE_MAXLOT))
{
LotSize = MarketInfo(Symbol(),MODE_MAXLOT);
}
if(MarketInfo(Symbol(),MODE_LOTSTEP) == 0.1)
{
LotSize = NormalizeDouble(LotSize,1);
}
else LotSize = NormalizeDouble(LotSize,2);
```

Akan ada sedikit perbedaan dalam cara mengukur ukuran lot dan fungsi memverikasi kode di awal akan ditambahkan. Karena level StopLoss telah diketahui sebelumnya, maka ini adalah tempat yang bagus untuk meletakkannya. Kode yang tersisa adalah rutinitas pesanan pasar modifikasi kami ;

```
// Buy Order
if(FastMA > SlowMA && BuyTicket == 0)
{
// Close Order
OrderSelect(SellTicket,SELECT_BY_TICKET);
if(OrderCloseTime() == 0 && SellTicket > 0)
        {
        double CloseLots = OrderLots();
```

```
            while(IsTradeContextBusy()) Sleep(10);
            RefreshRates();
            double ClosePrice = Ask;
            bool Closed =
OrderClose(SellTicket,CloseLots,ClosePrice,UseSlippage,Red);
            // Error handling
            if(Closed == false)
            {
            ErrorCode = GetLastError();
            string ErrDesc = ErrorDescription(ErrorCode);
            string ErrAlert = StringConcatenate("Close Sell Order - Error ",
            ErrorCode,": ",ErrDesc);
            Alert(ErrAlert);
            string ErrLog = StringConcatenate("Ask: ",Ask," Lots: ",LotSize,
            " Ticket: ",SellTicket);
            Print(ErrLog);
            }
}
// Open buy order
while(IsTradeContextBusy()) Sleep(10);
RefreshRates();
BuyTicket = OrderSend(Symbol(),OP_BUY,LotSize,Ask,UseSlippage,0,0,
```

```
"Buy Order",MagicNumber,0,Green);
// Error handling
if(BuyTicket == -1)
        {
        ErrorCode = GetLastError();
         ErrDesc = ErrorDescription(ErrorCode);
        ErrAlert = StringConcatenate("Open Buy Order - Error ",
        ErrorCode,": ",ErrDesc);
         Alert(ErrAlert);
          ErrLog = StringConcatenate("Ask: ",Ask," Lots: ",LotSize);
         Print(ErrLog);
        }
// Order modification
else
{
 OrderSelect(BuyTicket,SELECT_BY_TICKET);
 double OpenPrice = OrderOpenPrice();
// Calculate stop level
double StopLevel = MarketInfo(Symbol(),MODE_STOPLEVEL) * Point;
 RefreshRates();
 double UpperStopLevel = Ask + StopLevel;
 double LowerStopLevel = Bid - StopLevel;
```

```
 double MinStop = 5 * UsePoint;
 // Calculate stop loss and take profit
 if(StopLoss > 0) double BuyStopLoss = OpenPrice - (StopLoss * UsePoint);
 if(TakeProfit > 0) double BuyTakeProfit = OpenPrice + (TakeProfit *
UsePoint);
// Verify stop loss and take profit
 if(BuyStopLoss > 0 && BuyStopLoss > LowerStopLevel)
 {
 BuyStopLoss = LowerStopLevel - MinStop;
 }
 if(BuyTakeProfit > 0 && BuyTakeProfit < UpperStopLevel)
 {
BuyTakeProfit = UpperStopLevel + MinStop;
 }
// Modify order
 if(IsTradeContextBusy()) Sleep(10);
 if(BuyStopLoss > 0 || BuyTakeProfit > 0)
{
bool TicketMod = OrderModify(BuyTicket,OpenPrice,BuyStopLoss,
BuyTakeProfit,0);
// Error handling
if(TicketMod == false)
```

```
                    {
                    ErrorCode = GetLastError();
                    ErrDesc = ErrorDescription(ErrorCode);
                     ErrAlert = StringConcatenate("Modify Buy Order - Error ",
                    ErrorCode,": ",ErrDesc);
                     Alert(ErrAlert);
                     ErrLog = StringConcatenate("Ask: ",Ask," Bid: ",Bid," Ticket: ",
                    BuyTicket," Stop: ",BuyStopLoss," Profit: ",BuyTakeProfit);
                    Print(ErrLog);
                     }
            }
}
SellTicket = 0;
}
```

Sisa kode kami berisi blok penempatan pesanan pasar penjualan, serta fungsi **PipPoint()** dan **GetSlippage().** Anda dapat melihat kode lengkapnya pada Apendix B.

Sebagai catatan, kami menambahkan fungsi **IsTradeContextBusy** sebelum trade dilakukan. Kami ingin memastikan bahwa TradeThread tidak sedang digunakan sebelum melakukan trade. Kami menggunakan fungsi **RefreshRate()** sebelum variabel **Bid** dan **Ask,** untuk memastikan bahwa kami selalu menggunakan harga terkini.

Kami memulai dengan memilih tiket pesanan penjualan sebelumnya dan menutupnya menggunakan **OrderClose().** Jika fungsi gagal, blok penanganan

kesalahan dijalankan. Selanjutnya, kita membuka order pasar beli menggunakan **OrderSend().** Jika fungsi gagal, itu blok penanganan kesalahan dijalankan. Jika tidak, kami melanjutkan ke blok modifikasi pesanan.

Kami memilih pesanan yang baru saja ditempatkan menggunakan **OrderSelect(),** dan menetapkan harga pembukaan pesanan ke variabel **OpenPrice.** Kami kemudian menghitung level stop dan level stop atas dan bawah harga. Selanjutnya, kami menghitung harga StopLoss dan TakeProfit , setelahnya memverifikasi itu dan pada akhirnya kami memodifikasi pesanan menggunakan **OrderModify().** Blok penanganan kesalahan terakhir berkaitan dengan kesalahan dari pesanan modifikasi.

Berikut cara memodifikasi kode pada pesanan BuyStop ;

```
// Close order
OrderSelect(SellTicket,SELECT_BY_TICKET);
if(OrderCloseTime() == 0 && SellTicket > 0 && OrderType() == OP_SELL)
{
double CloseLots = OrderLots();
while(IsTradeContextBusy()) Sleep(10);
RefreshRates();
double ClosePrice = Ask;
bool Closed = OrderClose(SellTicket,CloseLots,ClosePrice,UseSlippage,Red);
// Error handling
 if(Closed == false)
        {
        ErrorCode = GetLastError();
```

```
            string ErrDesc = ErrorDescription(ErrorCode);
            string ErrAlert = StringConcatenate("Close Sell Order - Error
            ",ErrorCode,
            ": ",ErrDesc);
            Alert(ErrAlert);
            string ErrLog = StringConcatenate("Ask: ",Ask," Lots: ",LotSize,
            " Ticket: ",SellTicket);
            Print(ErrLog);
            }
        }
        // Delete order
        else if(OrderCloseTime() == 0 && SellTicket > 0 && OrderType() ==
OP_SELLSTOP)
        {
        bool Deleted = OrderDelete(SellTicket,Red);
        if(Deleted == true) SellTicket = 0;
        // Error handling
         if(Deleted == false)
            {
            ErrorCode = GetLastError();
            ErrDesc = ErrorDescription(ErrorCode);
            ErrAlert = StringConcatenate("Delete Sell Stop Order - Error
        ",ErrorCode,
```

```
            ": ",ErrDesc);
            Alert(ErrAlert);
            ErrLog = StringConcatenate("Ask: ",Ask," Ticket: ",SellTicket);
            Print(ErrLog);
            }
      }
```

Kami menambahkan kode menghapus pesanan menggunakan **OrderDelete**() setelah **OrderClose**(). Jenis pesanan dari pesanan jual sebelumnya menentukan fungsi mana yang digunakan untuk menutup pesanan.

Perbedaan utama antara kode berikut dan kode pesanan pasar adalah bahwa kami tidak memiliki blok modifikasi pesanan. Tidak perlu menempatkan StopLoss dan TakeProfit secara terpisah untuk pesanan tertuna. Karena itu kami akan menghitung StopLoss dan Take Profit sebelum melakukan pemesanan dengan **OrderSend().**

```
      // Calculate stop level
      double StopLevel = MarketInfo(Symbol(),MODE_STOPLEVEL) * Point;
      RefreshRates();
      double UpperStopLevel = Ask + StopLevel;
      double MinStop = 5 * UsePoint;
      // Calculate pending price
      double PendingPrice = High[0] + (PendingPips * UsePoint);
```

```
if(PendingPrice < UpperStopLevel) PendingPrice = UpperStopLevel +
MinStop;
// Calculate stop loss and take profit
if(StopLoss > 0) double BuyStopLoss = PendingPrice - (StopLoss * UsePoint);
if(TakeProfit > 0) double BuyTakeProfit = PendingPrice + (TakeProfit *
UsePoint);
// Verify stop loss and take profit
UpperStopLevel = PendingPrice + StopLevel;
double LowerStopLevel = PendingPrice – StopLevel;
if(BuyStopLoss > 0 && BuyStopLoss > LowerStopLevel)
{
BuyStopLoss = LowerStopLevel - MinStop;
}

if(BuyTakeProfit > 0 && BuyTakeProfit < UpperStopLevel)
{
BuyTakeProfit = UpperStopLevel + MinStop;
}
// Place pending order
if(IsTradeContextBusy()) Sleep(10);

BuyTicket =
OrderSend(Symbol(),OP_BUYSTOP,LotSize,PendingPrice,UseSlippage,
```

```
BuyStopLoss,BuyTakeProfit,"Buy Stop Order",MagicNumber,0,Green);

// Error handling

if(BuyTicket == -1)

{

 ErrorCode = GetLastError();

 ErrDesc = ErrorDescription(ErrorCode);


ErrAlert = StringConcatenate("Open Buy Stop Order - Error ",ErrorCode,

": ",ErrDesc);

 Alert(ErrAlert);

 ErrLog = StringConcatenate("Ask: ",Ask," Lots: ",LotSize," Price:
",PendingPrice,

" Stop: ",BuyStopLoss," Profit: ",BuyTakeProfit);

 Print(ErrLog);

 }

SellTicket = 0;
```

Pertama-tama, kita akan menghitung UpperStopLevel, kemudian kita akan menghitung dan mengklarifikasi harga pesanan yang tertunda yang tersimpan pada **PendingPrice**. Kemudian kita akan kembali menghitung **UpperStopLevel** dan menghitung **LowerStopLevel** sehingga mereka relatif terhadap harga pesanan tertunda. Perlu diingat bahwa kita tidak perlu menggunakan harga Ask atau Bid, atau angka dalam spread ketika memverifikasi harga StopLoss dan TakeProfit.

Pada akhirnya, kita akan menaruh pesanan menggunakan **OrderSend**(), beriringan dengan StopLoss dan TakeProfit. Kami memiliki fungsi penanganan

kesalahan standar untuk menangani kesalahan penempatan pesanan. Kode ini secara simpel memiliki fitur ekstra dalam menghitung dan memeverifikasi ukuran lot, StopLevel, StopLoss, TakeProfit, dan harga pesanan tertunda. Kami juga menambahkan TradeContextCheck dan kode penanganan eror.

Penggunaan Fungsi

Gagasan di balik pembuatan fungsi adalah membuat blok kode yang melakukan tugas yang sangat spesifik. Kode harus cukup fleksibel untuk digunakan kembali dalam berbagai situasi perdagangan. Setiap variabel atau kalkulasi eksternal perlu dilewatkan ke fungsi. Kami tidak dapat menganggap itu perlu nilai akan tersedia untuk fungsi kami jika tidak, karena fungsi tersebut dapat berada di file atau pustaka eksternal.

Agar konsisten, kami akan menyimpan nama yang sama untuk setiap variabel eksternal yang telah kami gunakan sejauh ini. Kami akan mengawali variabel-variabel ini dengan **"arg"**, untuk menunjukkan bahwa mereka adalah argumen fungsi.

Fungsi Pengukuran Lot

Fungsi ini merupakan fungsi **CalcLotSize**(), dapat diperhatikan bahwa **DynamicLotSize, EquityPercent, StopLoss** dan **FixedLotSize** semuanya adalah fungsi argumen sekarang. Variabel eksternal dengan nama-nama ini masih ada di program kami, kami hanya akan meneruskannya ke fungsi sebagai argumen sekarang.

```
double CalcLotSize(bool argDynamicLotSize, double argEquityPercent,
double argStopLoss,
double argFixedLotSize)
{
if(argDynamicLotSize == true)
 {
 double RiskAmount = AccountEquity() * (argEquityPercent / 100);
```

```
double TickValue = MarketInfo(Symbol(),MODE_TICKVALUE);
if(Point == 0.001 || Point == 0.00001) TickValue *= 10;
double LotSize = (RiskAmount / argStopLoss) / TickValue;
}
else LotSize = argFixedLotSize;
return(LotSize);
}
```

Argumen fungsi akan ditebalkan dan kodenya identik dengan kode perhitungan ukuran lot. Kami juga menambahkan statemen **return** pada akhir fungsi, yang berfungsi untuk mengembalikan nilai **LotSize**.

Fungsi itu sendiri akan ditempatkan di suatu tempat di file program kami, di luar fungsi **start ()** dan **init (),** atau itu akan terletak di file include eksternal. Dalam kasus terakhir, pernyataan **#include** di bagian atas program akan menyertakan file untuk digunakan dalam program kami.

Berikut cara menggunakan kode pada fungsi ini, pertama, kita akan mendata variabel eksternal yang akan digunakan untuk setting ukuran lot.

```
extern bool DynamicLotSize = true;
extern double EquityPercent = 2;
extern double FixedLotSize = 0.1;
extern double StopLoss = 50;
```

dan berikut merupakan perhitungannya, garis kode akan ditaruh dalam fungsi **start()**

double LotSize =

CalcLotSize(DynamicStopLoss,EquityPercent,StopLoss,FixedLotSize);

Variabel eksternal kami diteruskan ke fungsi sebagai argumen. Fungsi akan menghitung ukuran lot kami, dan nilainya akan disimpan dalam variabel **LotSize.** Perhatikan bahwa variabel ini berbeda dari

Variabel **LotSize** yang ada di dalam fungsi **CalcLotSize ().** Kedua variabel bersifat lokal untuk fungsinya, jadi meskipun mereka memiliki nama yang sama, mereka bukan variabel yang sama.

Memverifikasi Fungsi Lot

Ini akan menjadi fungsi yang terpisah, jika Anda memutuskan untuk menggunakan metode alternatif dalam menghitung ukuran lot. Terlepas dari metode penentuan ukuran lot, Anda harus memverifikasinya sebelum menggunakannya dengan fungsi penempatan pesanan.

```
double VerifyLotSize(double argLotSize)
{
 if(argLotSize < MarketInfo(Symbol(),MODE_MINLOT))
{
 argLotSize = MarketInfo(Symbol(),MODE_MINLOT);
 }
 else if(argLotSize > MarketInfo(Symbol(),MODE_MAXLOT))
 {
 argLotSize = MarketInfo(Symbol(),MODE_MAXLOT);
 }
```

```
if(MarketInfo(Symbol(),MODE_LOTSTEP) == 0.1)
{
argLotSize = NormalizeDouble(argLotSize,1);
}
else argLotSize = NormalizeDouble(argLotSize,2);
return(argLotSize);
}
```

Untuk fungsi ini, kami akan meneruskan variabel dengan ukuran lot yang kami hitung menggunakan **CalcLotSize()** sebagai argumen. Variabel argumen **argLotSize** kemudian diproses dan dikembalikan ke pemanggilan fungsi.

Fungsi Penempatan Pesanan

Sekarang saatnya untuk merakit fungsi penempatan pesanan pasar pembelian kami. Akan ada beberapa perbedaan antara fungsi penempatan pesanan kami dan kode yang kami ulas sebelumnya. kita tidak akan menutup pesanan dalam fungsi penempatan pesanan kami. Kami akan menangani penutupan pesanan secara terpisah.

Kami akan membuat fungsi untuk menutup pesanan di bab berikutnya. Sekarang saatnya untuk merakit fungsi penempatan pesanan pasar pembelian kami. Akan ada beberapa perbedaan antara fungsi penempatan pesanan kami dan kode yang kami ulas sebelumnya. Untuk satu, kita tidak akan menutup pesanan dalam fungsi penempatan pesanan kami. Kami akan menangani penutupan pesanan secara terpisah. Kami akan membuat fungsi untuk menutup pesanan di bab berikutnya.

Kami juga akan menghitung dan memodifikasi harga StopLoss kami dan TakeProfit keuntungan di luar fungsi penempatan pesanan. Karena ada beberapa cara

menghitung berhenti, kami harus tetap melakukan pemesanan fungsi penempatan sefleksibel mungkin, dan tidak mengikatnya dengan metode penghitungan berhenti yang telah ditentukan sebelumnya. Kode modifikasi pesanan telah dipindahkan ke fungsi terpisah.

Kami akan menempatkan pesanan pembelian kami pada harga pasar saat ini menggunakan **OrderSend(),** dan jika pesanan tidak dilakukan, kami akan menjalankan kode penanganan kesalahan. Dalam hal apa pun, kami akan mengembalikan nomor tiket ke fungsi panggilan, atau **-1** jika pesanan tidak dilakukan.

Kami menetapkan simbol pesanan menggunakan argumen **argSymbol,** bukan hanya menggunakan simbol grafik saat ini. Dengan cara ini, jika Anda memutuskan untuk melakukan pemesanan pada simbol lain, Anda dapat melakukannya dengan mudah. Daripada menggunakan variabel Bid dan Ask yang telah ditentukan, kita harus menggunakan **MarketInfo()** berfungsi dengan parameter **MODE_ASK** dan **MODE_BID** untuk mengambil harga **Bid** dan **Ask** untuk simbol tertentu.

```
int OpenBuyOrder(string argSymbol, double argLotSize, double argSlippage,

double argMagicNumber, string argComment = "Buy Order")

{

while(IsTradeContextBusy()) Sleep(10);

// Place Buy Order

 int Ticket =

OrderSend(argSymbol,OP_BUY,argLotSize,MarketInfo(argSymbol,MODE_ASK),

argSlippage,0,0,argComment,argMagicNumber,0,Green);

// Error Handling

 if(Ticket == -1)
```

```
{
 int ErrorCode = GetLastError();
 string ErrDesc = ErrorDescription(ErrorCode);
string ErrAlert = StringConcatenate("Open Buy Order – Error ",
ErrorCode,": ",ErrDesc);
Alert(ErrAlert);
string ErrLog = StringConcatenate("Bid:
",MarketInfo(argSymbol,MODE_BID),
" Ask: ",MarketInfo(argSymbol,MODE_ASK)," Lots: ",argLotSize);
Print(ErrLog);
}
return(Ticket);
}
```

Dalam fungsi **OrderSend (),** perhatikan bahwa kami telah menggunakan fungsi **MarketInfo ()** dengan parameter **MODE_ASK**, sebagai ganti variabel Ask yang telah ditentukan. Ini akan mengambil harga Permintaan saat ini untuk simbol mata uang yang ditunjukkan oleh **argSymbol.**

Jika perdagangan tidak berhasil dilakukan, rutinitas penanganan kesalahan akan berjalan. Kalau tidak, pesanan tiket akan dikembalikan ke fungsi panggilan, atau **-1** jika pesanan tidak dilakukan. Pesanan lengkap fungsi penempatan untuk pesanan pasar penjualan ada di AppendixD.

Penempatan Pesanan Tertunda

Untuk menempatkan pesanan yang tertunda, kami harus memberikan parameter untuk harga pesanan yang tertunda serta waktu kedaluwarsa pesanan. Argumen **argPendingPrice** dan **argExpiration** akan ditambahkan ke fungsi.

Kami akan menganggap bahwa harga pesanan yang tertunda, serta stop loss dan take profit, telah dihitung dan diverifikasi sebelum memanggil fungsi ini. Fungsi penempatan pesanan yang tertunda akan ditempatkan StopLoss dan TakeProfit dengan pesanan tertunda, jadi tidak ada fungsi modifikasi pesanan terpisah yang dibutuhkan.

Kami akan menganggap bahwa harga pesanan yang tertunda, serta StopLoss dan TakeProfit, telah dihitung dan diverifikasi sebelum memanggil fungsi ini. Fungsi penempatan pesanan yang tertunda akan ditempatkan StopLoss dan TakeProfit dengan pesanan tertunda, jadi tidak ada fungsi modifikasi pesanan terpisah yang dibutuhkan.

```
int OpenBuyStopOrder(string argSymbol, double argLotSize, double
argPendingPrice,

double argStopLoss, double argTakeProfit, double argSlippage, double
argMagicNumber,

datetime argExpiration = 0, string argComment = "Buy Stop Order")

{

while(IsTradeContextBusy()) Sleep(10);

// Place Buy Stop Order

int Ticket = OrderSend(argSymbol,OP_BUYSTOP,argLotSize,argPendingPrice,

argSlippage,argStopLoss,argTakeProfit,argComment,argMagicNumber,

argExpiration,Green);
```

```
// Error Handling
 if(Ticket == -1)
 {
 int ErrorCode = GetLastError();
 string ErrDesc = ErrorDescription(ErrorCode);
string ErrAlert = StringConcatenate("Open Buy Stop Order - Error
",ErrorCode,
": ",ErrDesc);
Alert(ErrAlert);
string ErrLog = StringConcatenate("Ask:
",MarketInfo(argSymbol,MODE_ASK),
" Lots: ",argLotSize," Price: ",argPendingPrice," Stop: ",argStopLoss,
" Profit: ",argTakeProfit," Expiration: ",TimeToStr(argExpiration));
Print(ErrLog);
}
 return(Ticket);
}
```

Perhatikan bahwa kami telah menetapkan nilai default 0 untuk **argExpiration**. Jika Anda tidak menggunakan pending memesan waktu kedaluwarsa, dan Anda ingin menggunakan komentar pesanan default, Anda cukup menghilangkan argumen untuk **argExpiration** dan **argComment** saat memanggil fungsi. Contoh berikut akan menempatkan pesanan penghentian pembelian tanpa waktu kedaluwarsa dan komentar pesanan default, "Buy Stop Order"

int Ticket =

OpenBuyStopOrder(Symbol(),LotSize,PendingPrice,StopLoss,TakeProfit,

UseSlippage,MagicNumber);

Kami telah menambahkan harga yang tertunda ke log di fungsi penanganan kesalahan kami, serta waktu kedaluwarsa, jika ada yang ditentukan. Fungsi **TimeToStr()** mengubah variabel datetime menjadi string yang dapat dibaca format.

Fungsi untuk membuka stop jual, limit beli, dan order limit penjualan identik dengan yang ini. Satu-satunya perbedaan adalah bahwa parameter tipe pesanan untuk fungsi **OrderSend()** yang diubah. Kamu dapat melihat semua fungsi penempatan pesanan dalam **AppendixD.**

Fungsi Penutupan Pesanan

Pada sesi ini, kita akan menerangkan cara menutup beberapa pesanan dengan metode yang lebih mudah.

bool CloseBuyOrder(string argSymbol, int argCloseTicket, double argSlippage)

{

OrderSelect(argCloseTicket,SELECT_BY_TICKET);

if(OrderCloseTime() == 0)

{

double CloseLots = OrderLots();

while(IsTradeContextBusy()) Sleep(10);

double ClosePrice = MarketInfo(argSymbol,MODE_ASK);

```
bool Closed =
OrderClose(argCloseTicket,CloseLots,ClosePrice,argSlippage,Red);
if(Closed == false)
{
int ErrorCode = GetLastError();
string ErrDesc = ErrorDescription(ErrorCode);
string ErrAlert = StringConcatenate("Close Buy Order - Error: ",ErrorCode,
": ",ErrDesc);
Alert(ErrAlert);
string ErrLog = StringConcatenate("Ticket: ",argCloseTicket," Ask: ",
MarketInfo(argSymbol,MODE_ASK));
Print(ErrLog);
}
}
return(Closed);
}
```

Pada variabel **ClosePrice,** kami menggunakan **MarketInfo()** untuk mengembalikan harga Ask pada indikasi mata uang. Kami menggunakan argument **argCloseTicket** and **argSlippage** pada penutupan tiket pesanan dan SlipPage. Apabila pesanan tersebut tidak berhasil ditutup, maka kami akan menjalankan blok penanganan eror, yang akan menampilkan nomor tiket dan harga tiket Ask pada log.

Kode untuk menjual pesanan akan identik, kecuali menggunakan harga Bid pada variabel **ClosePrice**. Anda dapat fungsi menjual pesanan pasar di AppendixD.

Fungsi Pesanan yang Tertunda

Berikut fungsi single pesanan tertunda. Ini akan berfungsi pada semua pesanan tertunda, buy atau sell.

```
bool ClosePendingOrder(string argSymbol, int argCloseTicket, double
argSlippage)
{
OrderSelect(argCloseTicket,SELECT_BY_TICKET);
if(OrderCloseTime() == 0)
{
while(IsTradeContextBusy()) Sleep(10);
bool Deleted = OrderDelete(argCloseTicket,Red);
if(Deleted == false)
{
int ErrorCode = GetLastError();
string ErrDesc = ErrorDescription(ErrorCode);
string ErrAlert = StringConcatenate("Close Pending Order - Error: ",
ErrorCode,": ",ErrDesc);
Alert(ErrAlert);
string ErrLog = StringConcatenate("Ticket: ",argCloseTicket,
```

```
" Bid: ",MarketInfo(argSymbol,MODE_BID),

" Ask: ",MarketInfo(argSymbol,MODE_ASK));

Print(ErrLog);

}

}

return(Deleted);

}
```

Fungsi Perhitungan StopLoss dan TakeProfit

Berikut cara memperhitungkan StopLoss dan TakeProfit secara singkat, Kami akan meneruskan variabel eksternal kami yang menunjukkan StopLoss atau TakeProfit dalam pips ke fungsi kami, serta harga pembukaan pesanan. Nilai pengembalian fungsi kami akan menjadi harga StopLoss dan TakeProfit. Berikut fungsi cara menghitung BuyStopLoss dalam pips ;

```
double CalcBuyStopLoss(string argSymbol, int argStopLoss, double argOpenPrice)

{

if(argStopLoss == 0) return(0);

double BuyStopLoss = argOpenPrice - (argStopLoss * PipPoint(argSymbol));

return(BuyStopLoss);

}
```

Pertama, kami akan memeriksa untuk melihat apakah tingkat StopLoss yang valid telah diteruskan bersama dengan fungsinya. Jika argumen **argStopLoss** adalah

0, maka kami mengembalikan nilai 0 ke fungsi panggilan, yang menunjukkan bahwa tidak ada stop loss telah ditentukan.

Selanjutnya, kami menghitung StopLoss dengan mengurangi StopLoss dalam pips dari harga pembukaan pesanan. Kami mengalikan **argStopLoss** dengan **PipPoint ()** untuk menghitung nilai fraksional, dan mengurangi dari **argOpenPrice**. Kami akan menggunakan harga Penawaran atau Permintaan (untuk pesanan pasar) atau tertunda yang dimaksud harga pesanan.

Perhatikan bahwa kami tidak memeriksa level stop atau memverifikasi bahwa StopLoss valid. Kami akan menggunakan serangkaian fungsi lain untuk memverifikasi atau menyesuaikan harga StopLoss. Anda tentu saja dengan mudah memodifikasi fungsi ini untuk memverifikasi harga StopLoss, menampilkan pesan kesalahan, atau secara otomatis sesuaikan harganya.

Berikut fungsi BuyTakeProfit pada pips :

```
double CalcBuyTakeProfit(string argSymbol, int argTakeProfit, double argOpenPrice)

{

if(argTakeProfit == 0) return(0);

double BuyTakeProfit = OpenPrice + (argTakeProfit * PipPoint(argSymbol));

return(BuyTakeProfit);

}
```

Fungsi untuk menghitung StopLoss dan TakeProfit untuk order penjualan tercantum dalam AppendixD. Perhatikan bahwa fungsi untuk menghitung SellStopLoss hampir identik dengan yang di atas untuk menghitung BuyTakeProfit, dan juga untuk BuyStopLoss dan SellTakeProfit.

Menghitung Level Verifikasi

Kami akan membuat dua set fungsi untuk menghitung dan memverifikasi level berhenti. Yang pertama hanya akan menghitung level stop di atas atau di bawah harga yang ditentukan, dan mengembalikan nilai boolean yang menunjukkan apakah harga yang ditunjukkan di dalam atau di luar level stop. Set fungsi kedua akan secara otomatis menyesuaikan harga sehingga berada di luar level stop, plus atau minus sejumlah tertentu pips.

Fungsi berikut memverifikasi apakah harga berada di atas level berhenti atas (harga pembukaan pesanan ditambah tingkat berhenti). Jika demikian, fungsi mengembalikan True, jika tidak False:

```
bool VerifyUpperStopLevel(string argSymbol, double argVerifyPrice,

double argOpenPrice = 0)

{

 double StopLevel = MarketInfo(argSymbol,MODE_STOPLEVEL) * Point;

 if(argOpenPrice == 0) double OpenPrice =
MarketInfo(argSymbol,MODE_ASK);

else OpenPrice = argOpenPrice;

double UpperStopLevel = OpenPrice + StopLevel;

 if(argVerifyPrice > UpperStopLevel) bool StopVerify = true;

else StopVerify = false;



 return(StopVerify);

 }
```

Kami melewati simbol mata uang, harga untuk memverifikasi, dan harga pembukaan pesanan (opsional) sebagai argumen. Secara default, level stop dihitung relatif terhadap harga Ask. Jika argOpenPrice adalah ditentukan, level stop akan dihitung relatif terhadap harga itu sebagai gantinya. (Gunakan ini saat memverifikasi berhenti kehilangan dan ambil harga untung dari pending order).

Fungsi akan memeriksa untuk melihat apakah **argVerifyPrice** lebih besar dari **UpperStopLevel**. Jika ya, nilai baliknya akan benar. Kalau tidak, salah. Anda dapat menggunakan fungsi ini untuk memeriksa StopLoss yang valid, ambil untung atau harga pesanan tertunda, tanpa mengubah harga aslinya. Berikut adalah contoh di mana kami memeriksa harga StopLoss dan menampilkan pesan kesalahan jika harga tidak valid:

```
bool Verified = VerifyUpperStopLevel(Symbol(),SellStopLoss);

if(Verified == false) Alert("Sell stop loss is invalid!");
```

Kode untuk memeriksa level stop di bawah harga saat ini atau yang tertunda ada di AppendixD. Fungsi kedua kami mirip, kecuali bahwa mereka akan secara otomatis menyesuaikan StopLoss yang tidak valid, TakeProfit atau harga pesanan tertunda ke yang valid:

```
double AdjustAboveStopLevel(string argSymbol, double argAdjustPrice, int
argAddPips = 0,

double argOpenPrice = 0)

 {

 double StopLevel = MarketInfo(argSymbol,MODE_STOPLEVEL) * Point;


 if(argOpenPrice == 0) double OpenPrice =
MarketInfo(argSymbol,MODE_ASK);
```

```
        else OpenPrice = argOpenPrice;
        double UpperStopLevel = OpenPrice + StopLevel;
        if(argAdjustPrice <= UpperStopLevel)
        {
        double AdjustedPrice = UpperStopLevel + (argAddPips *
PipPoint(argSymbol));
        }
        else AdjustedPrice = argAdjustPrice;
        return(AdjustedPrice);
        }
```

Argumen a**rgAdjustPrice** adalah harga yang akan kami verifikasi dan sesuaikan jika itu tidak valid. Kami telah menambahkan parameter opsional baru, **argAddPips**. Ini akan menambahkan jumlah pips yang ditentukan ke tingkat berhenti harga saat menyesuaikan harga yang tidak valid.

Seperti sebelumnya, kami menghitung StopLevel yang relatif terhadap harga Ask atau parameter **argOpenPrice**. Jika parameter a**rgAdjustPrice** berada di dalam level stop (mis. Tidak valid), harganya akan menjadi disesuaikan sehingga berada di luar level stop dengan jumlah pips yang ditentukan oleh **argAddPips**.

Jika harga yang ditentukan oleh **argAdjustPrice** valid, harga itu akan dikembalikan ke fungsi panggilan. Bagaimanapun, nilai pengembaliannya adalah yang ingin anda gunakan untuk TakeProfit anda, StopLoss atau harga pesanan tertunda. Kami akan menggunakan fungsi-fungsi ini dalam buku ini untuk memverifikasi level berhenti dan menyesuaikan harga kami.

Menambahkan StopLoss dan TakeProfit

Sesuai dengan ide kami untuk menjaga fungsi tetap fokus pada tugas-tugas sederhana dan diskrit, kami telah memindahkan modifikasi pesanan kami ke fungsi terpisah. Fungsi ini akan menambah atau memodifikasi StopLoss dan TakeProfit dari pesanan yang ditentukan. Kami akan menganggap stop loss dan harga untung telah dihitung dan diverifikasi,

```
bool AddStopProfit(int argTicket, double argStopLoss, double argTakeProfit)

{

if(argStopLoss == 0 && argTakeProfit == 0) return(false);

OrderSelect(argTicket,SELECT_BY_TICKET);

double OpenPrice = OrderOpenPrice();

while(IsTradeContextBusy()) Sleep(10);

// Modify Order

bool TicketMod =
OrderModify(argTicket,OrderOpenPrice(),argStopLoss,argTakeProfit,0);

 // Error Handling

if(TicketMod == false)

{

int ErrorCode = GetLastError();

string ErrDesc = ErrorDescription(ErrorCode);

string ErrAlert = StringConcatenate("Add Stop/Profit - Error ",ErrorCode,": ",

ErrDesc);

Alert(ErrAlert);
```

```
string ErrLog = StringConcatenate("Bid:
",MarketInfo(OrderSymbol(),MODE_BID),
" Ask: ",MarketInfo(OrderSymbol(),MODE_ASK)," Ticket: ",argTicket,
" Stop: ",argStopLoss," Profit: ",argTakeProfit);
Print(ErrLog);
 }
 return(TicketMod);
}
```

Kami memeriksa dulu untuk melihat apakah harga StopLoss atau TakeProfit telah disediakan. Jika tidak, kami akan keluar dari fungsinya. Jika tidak, kami akan memodifikasi pesanan menggunakan StopLoss dan TakeProfityang telah berlalu ke fungsi. Fungsi penanganan kesalahan akan berjalan jika modifikasi pesanan tidak berhasil. Ini fungsi akan bekerja pada semua jenis pesanan.