○ＭＱＬ５；翻訳まとめ「OnTrade()の使い方、他（その１）」　翻訳のみ実施　2012.10.12

・アメンボです、
　　久しぶりに「MQL5」の翻訳を再開いたします、
　　本稿の翻訳対象は「**OnTrade()**」関数です。

・簡潔に言えば、
「**OnTrade()**が呼ばれるのはTrade Event（トレード・イベント）が発生した場合」
なのですが、調べているうちに頭が混乱しました！（未だMQL5が判らない！）

注意；　・本資料は、まだＭＴ５での動作・検証を行っていません、
・本編は近々の検証用資料として、英文資料を意訳しながら纏めたもの（メモ）です。
訳した資料がある程度たまったところで、ＭＴ５をダウンロードして確認して
いくつもりです。・・・すいません、まだＭＴ５は使ったことが無いのです！
（実機で未検証の内容ですので、誤訳があるかもしれません）
・以上の状況を理解されたうえで、本稿内容を参照ください。
○本稿を「(その１)」としたのは、基本内容のみを記述したので、別の機会に応用や実施例等を
報告しようと考えているからです。

目次：

１．イベント「ハンドリング関数とトリガ」（現状の理解）

※表１；現時点での理解範囲で、全体と一応解説済みのものを整理してみます。

		EA;ExpertAdviser		Indicator	Script	
ﾊﾝﾄﾞﾘﾝｸﾞ関数	ｲﾍﾞﾝﾄ・ﾄﾘｶﾞ とﾓｰﾄﾞ別	関数使用	OrderSend 関数内発行	ｲﾝﾃﾞｨｹｰﾀ 表示	ｽｸﾘﾌﾟﾄ 実行	解説
OnStart()	—	—	○	—	○	？改めて解説必要
OnInit()	開始	○	—	○	—	済
OnDeinit()	終了	○	—	○	—	済
OnTick()	ﾃｨｯｸ	○	○	—	—	？改めて解説必要
	ﾏﾙﾁｶﾚﾝｼｰ･ﾓｰﾄﾞ	○	○	—	—	**未＜別途＞**
OnTimer()	ﾀｲﾏｰ	○	○	—	—	済
OnTrade()	order・deal・position	○	？	—	—	**本稿**
OnTester()	ｽﾄﾗﾃｼﾞｰ・ﾃｽﾀｰ	○	—	—	—	済
OnBookEvent()	板(DOM)情報	○	○	—	—	済
OnChartEvent()	未確認	○	○	○	—	未＜別途＞
	ｶｽﾀﾑ･ｲﾍﾞﾝﾄ	○	○	○	—	**未＜別途＞**
OnCalculate()	ｲﾝﾃﾞｨｹｰﾀ表示計算	—	—	○	—	済；**半分残**
	簡略ﾀｲﾌﾟ	—	—	○	—	**未＜別途＞**

※DOM：Depth of Market　要するに「板情報」のこと　※「**青書**」部は、追加・修正した部分

※「OrderSend 関数内発行」とは、例えば、「OnTimer()」内で「OrderSend」発行が可能と言う意味で使いました。

２．基礎知識の再確認と、MQL5（MT5）特有の注意点

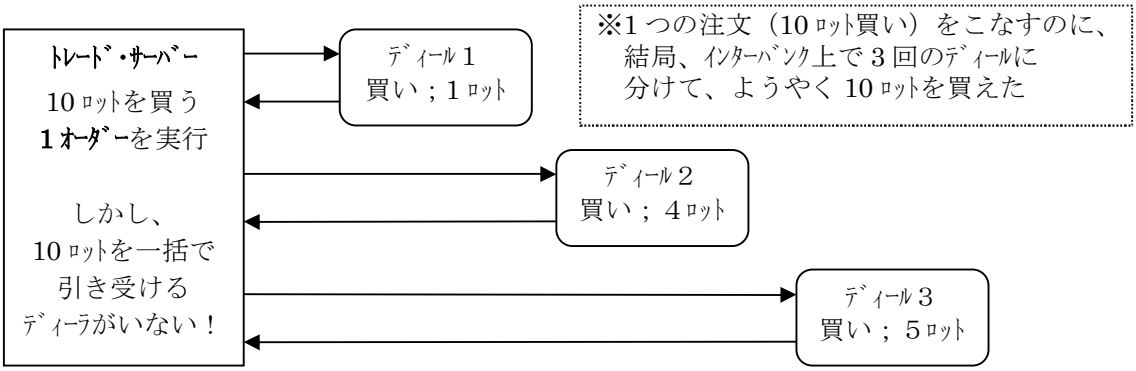
（１）Order、Deal、Position の違い

・アメンボは当初、MQL5 の資料をあたっていて「Order；オーダー」と「Deal；ディール」の違いが良く判りませんでした。（諸兄はどうですか？）簡潔に再確認を実施します。

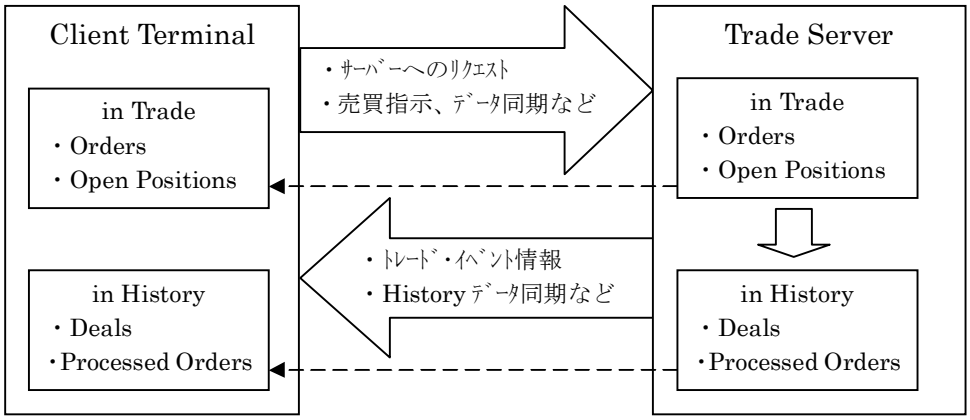
表２；

	解　説
Orders　ｵｰﾀﾞｰ	・「注文」ですよね、問題なし
Deals　ﾃﾞｨｰﾙ	・１つの注文に対して実行される内容のこと！ ・例えば、１つの Order（注文）で「10 ﾛｯﾄ」の買い注文を出したとき、「10 ﾛｯﾄ」を一度に売ってくれるﾃﾞｨｰﾗｰがおらず、「1 ﾛｯﾄ」→「4 ﾛｯﾄ」→「5 ﾛｯﾄ」と 3 人のﾃﾞｨｰﾗｰとの取引になった場合、1 回のｵｰﾀﾞｰが 3 回のﾃﾞｨｰﾙで処理されたことになります。（図１参照）
Positions ﾎﾟｼﾞｼｮﾝ	・言わずと知れた「建て玉」ですよね、問題なし ただし MT5 では同一通貨の両建てが出来ません。

図１；



（2）クライアント･ターミナルとトレード･サーバーの関係　図２



※トレード中の Order や Position データはサーバーが管理している。

※MQL5（ターミナル）がサーバーから情報を受け取るに際して、
キャッシュ(cache)を経由する場合が多々あるが、これについては別途解説予定です。

３．Trade Event の種類（OnTrade()が呼び出されるイベント）

（1）Trade Event（ﾄﾚｰﾄﾞ･ｲﾍﾞﾝﾄ）の発生原因

基本は、下記の「４つ」である。

- ・アクティブ・オーダー（注文）に変化があった
- ・ポジション（建て玉）に変化があった
- ・ディールが発生した
- ・トレード・ヒストリー内容に変化があった

※注意点；

１つのオペレーションが、<u>複数のトレード・イベント</u>を発生する場合が多い。

例；Pending Order があったとして、これがヒットした場合には、

①ディールが発生してトレード・ヒストリーに記録される

②次に、アクティブ・オーダーのリストから、この Pending Order が削除されて、
代わりに、ヒストリー・オーダーのリストに記録される

（2）呼出し例１；手動で発注した場合

・具体的にどの様な場合に OnTrade()が呼出されるのか？

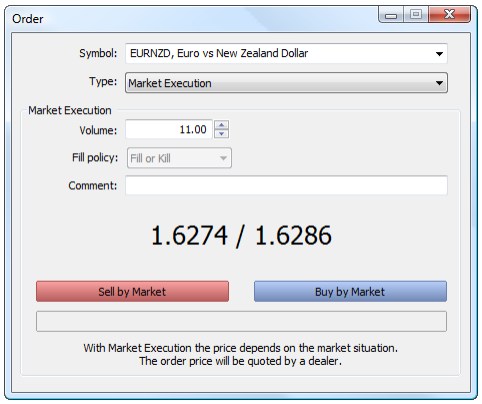
＜操作＞

１）まず、下記のようなコードのみを設定しておいて、

```
void OnTrade()
{
  Alert("Trade event occurred");
}
```

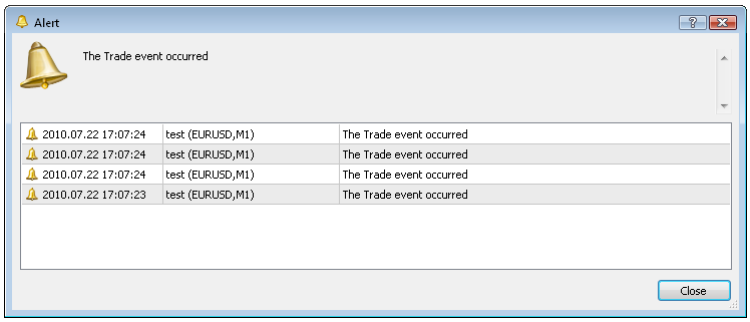
※中身は空の「OnInit()、OnDeinit()、OnTick()」はコード中に記載あり。

２）手動（マニュアル）で裁量トレードを実施する



ここで［Sell by Market］か
［Buy by Market］を選択する

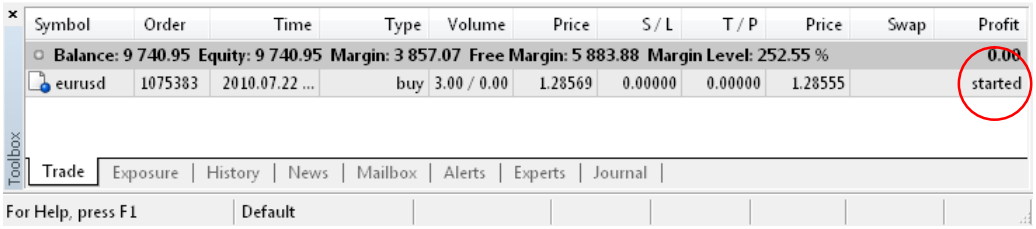
３）結果；　上記の「１)、２)」の組合せを実施すると、



上記の様に、最低「４回」OnTrade()が呼び出される。
その呼出しイベントは下記内容；

①オーダー（注文）の生成
②ディール（Deal）の実行・・１回以上
③オーダー（注文）が完結すると、ヒストリーに記述される
④ポジション（建玉）のオープン

※上記のプロセスの発生は、ＭＴ５の［Trade］タブで確認可能。
・「started」状態として発生したものが該当する

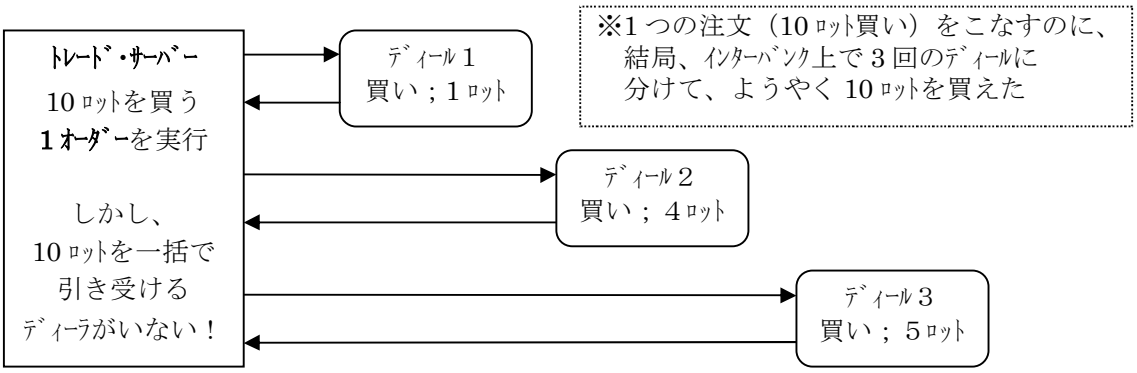


４）解析の第一歩

・OnTrade()の発生前後で、OrdersTotal()により Order（オーダー）の数を比較することから始めます。（概要は、本稿「４．（２）」を参照）
（具体例は別途解説予定です）

（３）呼出し例２；１つの Order が複数の Deal で処理される場合

・コードは記載せずに、「図１」を使って概念を解説



補足；この例では、各ディールごとに、引き受け手（ディーラー）が異なるわけです。

※上記の例では、果たして「何回」OnTrade()が呼出されるでしょうか？

ステップ１；(ディール１)
①「１ロット」の買いディール
②「ロット数」の変化；残　１０→９
ステップ２；(ディール２)
①「４ロット」の買いディール
②「ロット数」の変化；残　９→５
ステップ３；(ディール３)
①「４ロット」の買いディール
②「ロット数」の変化；残　５→０
③オーダーが完了したので、トレード・ヒストリーに記録

上記の各イベントごとに、OnTrade()が呼び出されます、
つまり、「**２回＋２回＋３回＝７回**」ほど OnTrade()が呼出されることになります！

４．OnTrade() の使い方（MQL5 コード基本構成）

（１）関数の概要

１）動作タイミング；Trade Event（ﾄﾚｰﾄﾞ･ｲﾍﾞﾝﾄ）が発生したときに call される　（呼ばれる）

２）引数；なし

３）戻り値型；void　・・要は戻り値は「なし」

４）関数；まとめ

関　数	引数	戻り値	機　能
void OnTrade();	なし	なし	ﾄﾚｰﾄﾞ･ｲﾍﾞﾝﾄ発生ごとに 呼び出される

（２）基本構成

１）基本形；Order、Deal、Position の何に変化があったかを調べ、必要な処理に移る

手順；　－１．何が OnTrade()を呼び出したのかを調べる（基本４項目）

①「オーダー」数に変化があるか？
②「ポジション」数に変化があるか？
③ヒストリー・データ中の「ディール」数に変化があるか？
④ヒストリー・データ中の「ポジション」数に変化があるか？

－２．各呼び出し原因ごとに対応した処理を行う

コード例（基本部分のみ）；※「キャッシュ」については、別途解説予定です。

```
・・・・・
int             orders;          //  現在有効な注文数（オーダー数）
int             positions;       //  建て玉数（ポジション数）
int             deals;           //  ﾄﾚｰﾄﾞ・ﾋｽﾄﾘｰ（ｷｬｯｼｭ）中のディール数
int             history_orders;  //  ﾄﾚｰﾄﾞ・ﾋｽﾄﾘｰ（ｷｬｯｼｭ）中の注文数（オーダー数）
・・・・・
//  ヒストリー・データの調査範囲を決めるパラメーター
datetime        start;           //  ｷｬｯｼｭにｺﾋﾟｰするﾄﾚｰﾄﾞ・ﾋｽﾄﾘｰの開始時点
datetime        end;             //  ｷｬｯｼｭにｺﾋﾟｰするﾄﾚｰﾄﾞ・ﾋｽﾄﾘｰの最終時点（通常は現時点）
//
input long my_magic=555;         //  position_ID
//  初期化処理
int OnInit()
{
  //  エラーのリセット
      ResetLastError();
  //
      end=TimeCurrent();
      start=＊＊＊＊             //  必要な値にセット
  //  ヒストリー・データをキャッシュ上にコピーする
      bool selected=HistorySelect(start,end);
  //  最新の各データ数を入手する
      orders=OrdersTotal();
      positions=PositionsTotal();
      deals=HistoryDealsTotal();
      history_orders=HistoryOrdersTotal();
  //
   return(0);
}
//  ＥＡ等の本体
void OnTick()
  {
  //  最新の各データ数に更新する
      orders=OrdersTotal();
      positions=PositionsTotal();
      deals=HistoryDealsTotal();
      history_orders=HistoryOrdersTotal();
  //・・・定常的な処理を記述・・・
      return;
  }
//  トレード・イベント発生時の処理
void OnTrade()
  {
   //++++++++++++++++++++++++++++++++++++
   //  何が OnTrade()を呼び出したのかを調べる
   //------------------------------------
   //１．ステップ１；
```

```
   //（１）ヒストリー・データ入手準備（キャッシュ・エリアにコピーする）
       start=＊＊＊;
       end=TimeCurrent();
       bool selected=HistorySelect(start,end);
        ・・・・・
   //（２）現在のデータを入手（ダイレクトに入手）
       orders=OrdersTotal();
       positions=PositionsTotal();
       deals=HistoryDealsTotal();
       history_orders=HistoryOrdersTotal();
        ・・・・・
   //--------------------------------------
   //２．ステップ２；
   //　現在有効な「オーダー、ポジション」数、及び
   //　ヒストリー中の「Dealディール、オーダー」数に
   //　変化があったか否かを調べていく。
   //----------
   //　①有効な「オーダー」数に変化があるか？
   if(curr_orders!=orders)
      {
       //　◎変化があったことを検出したら、
       /*
         ＜変化に伴う処理＞を記述する
       */
       //　データ更新
       orders=curr_orders;
      }
   //----------
   //　②「ポジション」数に変化があるか？
   if(curr_positions!=positions)
      {
       //　◎変化があったことを検出したら、
       /*
         ＜変化に伴う処理＞を記述する
       */
       //　データ更新
       positions=curr_positions;
      }
   //----------
   //　③ヒストリー・データ中の「ディール」数に変化があるか？
   if(curr_deals!=deals)
      {
       //　◎変化があったことを検出したら、
       /*
         ＜変化に伴う処理＞を記述する
       */
       //　データ更新
       deals=curr_deals;
      }
   //----------
   //　④ヒストリー・データ中の「ポジション」数に変化があるか？
   if(curr_history_orders!=history_orders)
      {
       //　◎変化があったことを検出したら、
       /*
         ＜変化に伴う処理＞を記述する
       */
       //　データ更新
      history_orders=curr_history_orders;
      }
  }
```

以　上