○ＭＱＬ５；翻訳まとめ　「８つのデータ構造体（その２）」　翻訳のみ実施 2013.08.18

副題；OrderSend()と OrderSendAsync()

・アメンボです、
今回は「その２」なのですが、MQL5 では MQL4 に比較すると、個々の関数や機能が一見複雑に絡み合って解きほぐすのに苦労します。
解きほぐしてみると、整然とした体系になっているのですが、結構複雑なので「適切な解説本が無い（たぶん）」現状では理解するのが大変です。

・と、言うような状況にもめげずに、解析を進めます。
ただ、理解のために、頻繁に横道に逸れる必要があることを、ご容赦ください。

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。・・・アメンボは、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。

目次：

１.「８つの定義済み（データ）構造体」の残り４つ

・MQL5 に定義済の、データを一纏めにして扱う構造体（即ちデータ構造体）は、全部で
「８個」あり、うち半分は前回（その１）で解説済、本稿で残りの「４個」解説します。（下表）

	データ構造体	配列扱い	簡単な解説	備考
1	MqlDateTime	無？？	日時データ（date and time を扱う）の構造体	済
2	MqlParam	有	IndicatorCreate()でｲﾝﾃﾞｨｹｰﾀを作成する際に、ﾊﾟﾗﾒｰﾀ設定に使用	済
3	MqlRates	有	ヒストリカル・データ（含 price、volume、spread）を扱う	済
4	MqlBookInfo	有	Depth of Market（板表示）の情報取得に使用する	済
5	MqlTradeRequest	無	注文（ｵｰﾀﾞｰ）内容を設定するために使用する	**本稿**
6	MqlTradeResult	無	確認時点での、注文した結果の内容をチェックするために使用する	**本稿**
7	MqlTradeTransaction	無	注文の処理過程での情報をチェックするために使用する	**本稿**
8	MqlTick	無	現在の prices 情報を迅速に収集するために使用する	**本稿**

２．まず、「MqlTick 構造体」を簡単に解説（１つ目）

（１）データ構造体定義

・本構造体は「為替ペア」の Tick データを扱う構造体ですが、MQL4 とは微妙に意味合いが
異なります。

```
struct MqlTick
  {
   datetime     time;          // Time of the last prices update
   double       bid;           // Current Bid price
   double       ask;           // Current Ask price
   double       last;          // Price of the last deal (Last)
   ulong        volume;        // Volume for the current Last price
  };
```

※各メンバーの意味には要注意です、２系統のデータが入っているように観えます。
（アメンボはチョット、自信なし）

要　素	内　容
bid、ask	現在の Tick データ（提示されている値）です、 直近の約定行為（deal）によって影響を受ける。
time、last、volume	最後（直近）に約定（deal）した、 time；時間、　last；売買価格、　volume；ボリューム

（２）使用（呼出し）手順

・MQL5 特有！と言うか、呼出す前に「２段階」の前準備が必要で、
その後にやっとデータを参照することが可能になります。
（　厳密なのかも知れないけれど、何でこんな仕様にしたんでしょうね？　）

①ステップ１；　MqlTick 構造体名; による構造体の宣言
②ステップ２；　SymbolInfoTick(Symbol(),構造体名)により構造体にデータをセットする
③ステップ３；　「構造体名.ask」等によるデータ呼出し

（３）使用例

```
void OnTick()
  {
   MqlTick last_tick;
//---
   if(SymbolInfoTick(Symbol(),last_tick))
     {
      Print(last_tick.time,": Bid = ",last_tick.bid,
            " Ask = ",last_tick.ask,"  Volume = ",last_tick.volume);
     }
   else Print("SymbolInfoTick() failed, error = ",GetLastError());
//---
  }
```

（４）補足＆参考；　※SymbolInfoTick()関数；

機能；　指定「為替ペア」の、現在「価格」と最新「約定時間」情報を MqlTick 構造体に設定します。

```
bool  SymbolInfoTick(
   string    symbol,      // 為替ペア名
   MqlTick&  tick         // (MqlTick) 構造体名へのポインタ参照 (&が付いているのに注意)
   );
```

パラメータ；　*symbol*　[in]　為替ペア名

tick　[out] 指定した MqlTick タイプ構造体に、現在「価格」と最新「約定時間」情報等が設定されます。

返し値；　・成功　→　true を返す。

・失敗　→　false を返す。

３．残り３個の構造体に入る前に、手短に予備知識を整理する

・ユーザーから観た場合の、大きな相違点のみを比較します。

（１）MT4 と MT5 の機能比較（ポイントのみ）

特徴	MetaTrader 4	MetaTrader 5
トレード		
ワン・クリック・トレード	○（ﾋﾞﾙﾄﾞ 482 以降）	○
板情報（Market Depth）	×	○
外部サーバーへの発注（Exchange Execution）	×	○
複数オーダー実行時（同一方向のポジション）	ポジションの個別管理可能	トータルでの管理となる、個別管理は出来ず
チャート・タイムフレーム	9	21
Market Analysis		
テクニカル・インディケータ	30	38
グラフィック・オブジェクト	31	44
Expert Advisor Programming Language	MQL4	MQL5

（２）MT5 に於ける、４つの注文実行（Execution）モード

・MT5 では、「Exchange Execution」が新たに導入されている。(提供の有無はブローカーによる)

注文実行 (Execution) モードの種類	MT4	MT5	内　容	備　考
Request Execution リクエスト注文	○	○	・成行注文可能ロット数を超えた場合にリクエスト注文になり、設定取引ロットでブローカーが取引可能な価格（レート）をリクエストし、その価格を確認した後で注文を実行する。	※１参照
Instant Execution インスタント注文 （ａ）	○	○	・表示される価格は、ブローカーから提示される価格である。明確な価格が提示されるが、ブローカーのチェックが入る可能性あり。 ・ＤＤ（ディーラーズ・デスク）のある業者での採用が多い	通常は、(a)か(b)の一方のみが提供される。
Market Execution マーケット注文 （ｂ）	○	○	・注文を直接に市場（インターバンク・マーケット）に流す。スリップ・ページは無いが表示される価格はあくまで目安で、市場で成立した価格が「約定値」になる。 ・ＮＤＤ（ノン・ディーラーズ・デスク）のある業者での採用が多い	
Exchange Execution ｲｸｽﾁｪﾝｼﾞ注文	×	○	・トレード処理を「**外部のトレード・システム**」で実行させることが出来る。トレードはマーケット「成行」価格で実行。 **※アメンボの勝手な解釈；** **たぶん、これを活用すると「両建」が可能になる、と思う！**	**MT5 で両建！？** ・具体的な使用例が見つからず
(Pending Order)	○	○	(従来通り)	(参考)

※通常、各ブローカーは「Instant Execution」か「Market Execution」のどちらか一方を提供。

※１；MT4 環境での「Request Execution」例・・・機能提供の有無は、ブローカーによる

・成行注文可能ロット数を超えた場合はリクエスト注文となり、
設定した取引量（ロット）にて取引可能なレートをリクエストする。

使用手順例；

１．通貨の確認
２．注文種別［価格提示リクエスト］選択
３．数量に「ロット数」を入力
４．［価格提示リクエスト］ボタンを選択

５．取引可能なレートが提示されるので、
確認後に、［新規取引：成行買い］
または［新規取引：成行売り］
ボタンより注文を送信する。

※リクエストにより提示されたレートは、
画面上で「カウントダウン」される
時間（数秒）以内でご注文を送信
する必要がある。

（３）OrderSend(..)と OrderSendAsync(..)、その書式と機能の違い

・Mql5 に於ける注文（リクエスト）発行は、下記の２つの関数を用いて行います。
一つは mql4 でも御馴染みの「OrderSend()」ですが、使い方は全く異なります。
一方、「OrderSendAsync()」と言う、一見すると良く判らない関数もあります。

・先ずは、その全体像を観てみます。
しかし、それにしても「何か（ここでは構造体)」を解説しようとすると、
イモズル式に解読・解説範囲が広がっていくのには、Mql5 の特徴ですかねえ！！（しんどい）

－１．概要

・OrderSend()と OrderSendAsync()は、共に「売買」注文（リクエスト）を発行するために
使用する関数です。

・注文（リクエスト）内容を構造体「MqlTradeRequest &request」に設定後に発行し、
その結果は、構造体「MqlTradeResult &result」でチェック（確認）出来ますが、
確定値を得るには、サーバーからの応答確認を行うなど必要な手続きを踏む必要があります。

・OrderSendAsync()は、特に高速トレード用に準備されており、OrderSend()とは使用手順
が異なるので注意する必要があります。

－２．「OrederSend()と OrderSendAsync()」の用途と速度比較

・「OrederSend()＝通常のトレード用」、「OrderSendAsync()＝高速トレード用」と、
理解して良さそうです。

・以下に、英語のフォーラムに掲載されていた「実行速度」比較記事を示します。
例；３回実行したときのデモサーバーでの所要時間（詳細は不明です、悪しからず）

	用途	？？	所要時間		
			１回目	２回目	３回目
OredrSend()	通常トレード用	1 buy limit	1000 ms	3000 ms	8978 ms
OrderSendAsync()	高速トレード用	10 buy limit	78 ms	94 ms	125 ms

？？OrdreSendAsync()の「10 buy limit」は「たぶん 10 回送信した」の意味だと思うの
だけれど、自信ナシ。（短すぎる時間は計測できないので、「10 buy」を使ったようだ）

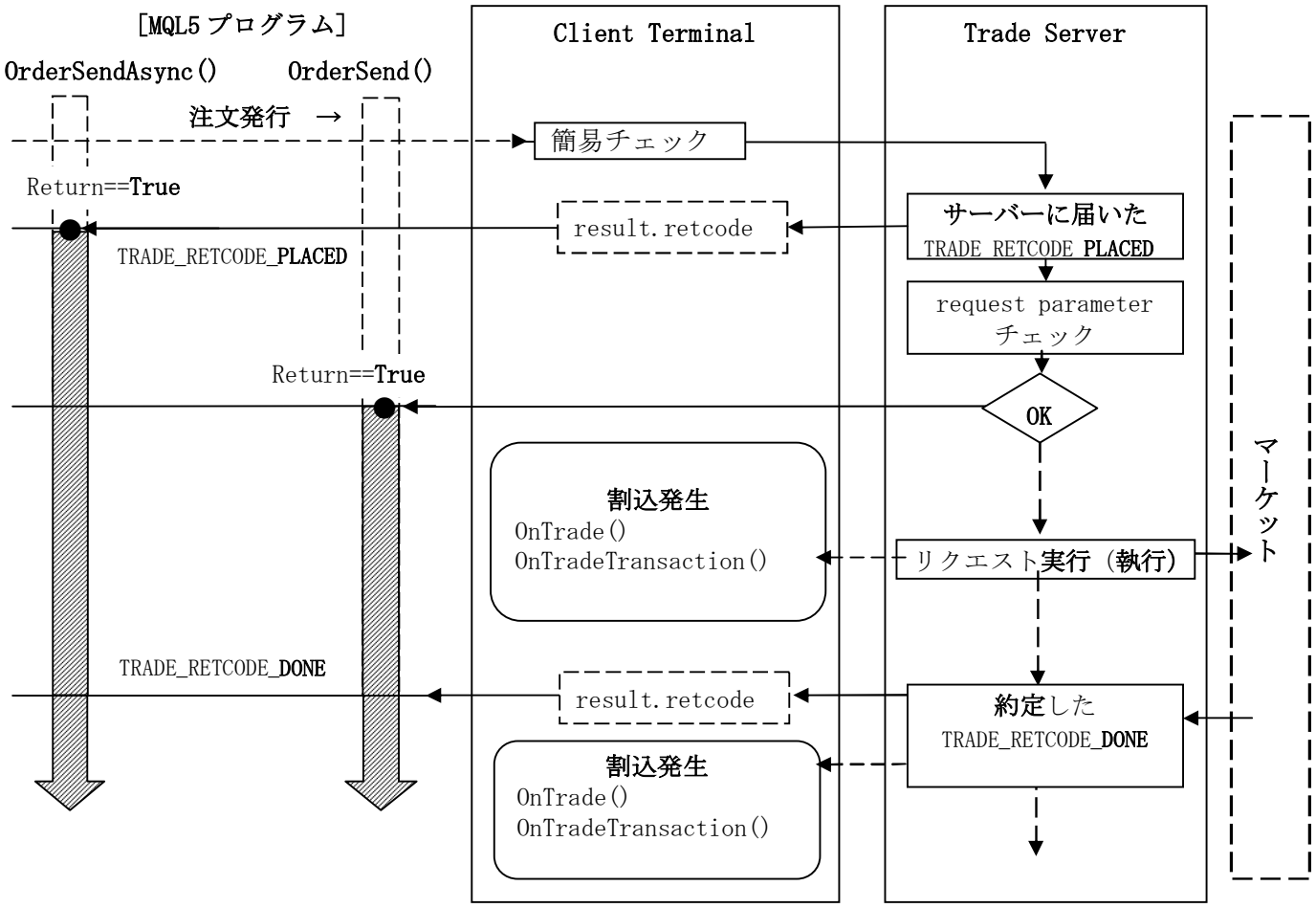
－３．書式と関連する構造体・関数

※各構造体の詳細内容は、「Ｐ８；４．「３個」の構造体」を参照。

	各、構造体の役割		
書　式	request	result	trans
bool OrderSend(MqlTradeRequest &request, MqlTradeResult &result);	注文の内容を 構造体各メンバー に設定する	注文結果の 確認時点での内容が 構造体各メンバー値 に反映される	—
bool OrderSendAsync(MqlTradeRequest &request, MqlTradeResult &result);	同上	同上	—
＜補足＞ void OnTradeTransaction(const MqlTradeTransaction &trans, const MqlTradeRequest &request, const MqlTradeResult &result)	割込を発生させた、 注文内容が構造体 各メンバーの 内容に反映される	割込を発生させた、 注文結果の 割込時点での内容が 構造体各メンバー値 に反映される	割込を発生させた、 注文プロセスの、各 トレード・イベントでの 詳細内容が構造体 各メンバーに反映される

－４．注文発行からトレード・サーバーまで（関係要素の相関図）

※Market Execution の場合で解説（チョット自信が在りません。MQL5 は複雑！です）



部で、次の「発行」が可能になる

※トレード・イベント結果の検出方法には、下記の２方法あり
① MqlTradeReques 構造体ﾒﾝﾊﾞｰの「retcode」で検出
②「割込発生」で検出、

※次の「リクエスト発行」が可能になるのは、「Return==True」後。
ただし、これは「リクエストが実行（執行）」されたことを
意味するものではナイことに注意。（上図参照）

※OredrSendAsync()では、「高速発注」が可能となる。

※要点捕捉；　「result.retcode」の使用例（ポイント）

```
struct MqlTradeResult
  {
     uint     retcode;         // Operation return code
   . . . . . .
  };

MqlTradeRequest result;
if(result.retcode==TRADE_RETCODE_DONE) //約定した
    {
      . . . . . .
    }
```

－５．「Return==True」とは何を意味するのか

※翻訳途中で「OrderSend()と OrderSendAsync()」の「Return==True」はどのような意味を持つのか、混乱したので、敢えて纏めることにしました。

※翻訳と類推から纏めているので、小生の誤解があれば、ご容赦ください。

	Return == TRUE であるとは「次の注文発行が可能と言うこと」で、
OrderSend() ※通常トレード用	・トレードサーバーがリクエストした構造体を受取り、パラメータ等のチェック結果がＯＫで、次の処理過程（プロセス）に進んだことを意味する。 ・トレードが成功裏に実行されたことを意味している訳では無い。
OrderSendAsync() ※高速トレード用	・とにかく、サーバーにリクエスト内容が「届けられた」（送られた）。 ・サーバーが「受付けること」は保証しない。 なにせ、パラメータ等のチェック結果がＯＫか否かは未だ不明なので。

－６．トレード結果・過程（プロセス）捕捉トリガ概要

・[注文（リクエスト）発行] ⇒ [トレード・プロセス進行]　を、捕捉するには、以下の表に示す「２通り」の方法があります。

トレード・イベント捕捉方法	コード記述例（一部、繰返しアリ）
MqlTradeResult 構造体メンバー『retcode』モニタによる捕捉	struct MqlTradeResult { uint retcode; // Operation return code ・・・・・・・・・・ }; コード（要点） MqlTradeRequest result; if(result.retcode==TRADE_RETCODE_DONE) //約定した { ・・・・・・・・ }
『割込み』による捕捉	OnTrade()による捕捉； ※ポジション、オーダー数等の変化により発生 「result」をモニターすることで、変化内容を確認できる。
	OnTradeTransaction()による捕捉； ※トレード・プロセスの進行により発生 「trans」をモニターすることで、変化内容を確認できる。

※トレード・プロセス；

・「OrderSend()　、又は OrderSendAsync()」により発生する「トレード・プロセス」例を、以下に示します。(各過程でトレード・イベントを発生します)

プロセス（処理過程）	トレード・プロセス（トリガ・イベント）例
ステップ１	handling a trade request　トレード・リクエストを受付・処理する
ステップ２	changing open orders　注文（オーダー）数を変更する
ステップ３	changing orders history　オーダー履歴を変更する
ステップ４	changing deals history　ディール履歴を変更する
ステップ５	changing positions　ポジション数を変更する

・各過程で、

①「result.retcode」の値は変化し、また　②「割込み」も発生していきます。

４．「３個」の構造体

※翻訳の手を抜きました、すいません。

英文内容は原文のまま載せました。（それほど難しい英文では無いですし！）

（１）MqlTradeRequest 構造体（２つ目）・・注文（リクエスト）内容を記述するのに使用

```
struct MqlTradeRequest
  {
   ENUM_TRADE_REQUEST_ACTIONS    action;   // Trade operation type
   ulong                         magic;    // Expert Advisor ID (magic number)
   ulong                         order;    // Order ticket
   string                        symbol;   // Trade symbol
   double                        volume;   // Requested volume for a deal in lots
   double                        price;    // Price
   double                        stoplimit;// StopLimit level of the order
   double                        sl;       // Stop Loss level of the order
   double                        tp;       // Take Profit level of the order
   ulong                         deviation;//
                                //Maximal possible deviation from the requested price
   ENUM_ORDER_TYPE               type;     // Order type
   ENUM_ORDER_TYPE_FILLING       type_filling;// Order execution type
   ENUM_ORDER_TYPE_TIME          type_time; // Order expiration type
   datetime                      expiration;// Order expiration time
                                     //(for the orders of ORDER_TIME_SPECIFIED type)
   string                        comment;  // Order comment
  };
```

－１．構造体メンバーの内容；

メンバー	内　容
action	Trade operation type. Can be one of the ENUM_TRADE_REQUEST_ACTIONS enumeration values.
magic	Expert Advisor ID. It allows organizing analytical processing of trade orders. Each Expert Advisor can set its own unique ID when sending a trade request.
order	Order ticket. It is used for modifying pending orders.
symbol	Symbol of the order. It is not necessary for order modification and position close operations.
volume	Requested order volume in lots. Note that the real volume of a deal will depend on the order execution type.
price	Price, reaching which the order must be executed. Market orders of symbols, whose execution type is "Market Execution" (SYMBOL_TRADE_EXECUTION_MARKET), of TRADE_ACTION_DEAL type, do not require specification of price.
stoplimit	The price value, at which the Limit pending order will be placed, when price reaches the *price* value (this condition is obligatory). Until then the pending order is not placed).
sl	Stop Loss price in case of the unfavorable price movement
tp	Take Profit price in the case of the favorable price movement
deviation	The maximal price deviation, specified in points
type	Order type. Can be one of the ENUM_ORDER_TYPE enumeration values.
type_filling	Order execution type. Can be one of the enumeration ENUM_ORDER_TYPE_FILLING values.
type_time	Order expiration type. Can be one of the enumeration ENUM_ORDER_TYPE_TIME values.
expiration	Order expiration time (for orders of ORDER_TIME_SPECIFIED type)
comment	Order comment

－２．注文実行（Execution）タイプと構造体メンバー

※注文実行（Execution）タイプにより、指定が必須のメンバー（項目）、指定が無くても良いメンバー（項目）がある様なので、以下に纏めます。

「○」；必須、　「△」；指定可能（無くてもＯＫ）、　「無印」；使用セズ

	action	magic	order	symbol	volume	price	stoplimit	sl	tp
Request Execution	○	△		○	○	○		○	○
Instant Execution	○	△		○	○	○？		○	○
Market Execution	○	△		○	○				
Exchange Execution	○	△		○	○				
SL & TP Modification	○			○				○	○
Pending Order	○	△		○	○	○	○	○	○
Modify Pending Order	○		○			○		○	○
Delete Pending Order	○		○						

上表に続く、

	deviation	type	**type filling**	type time	expiration	comment
Request Execution	○	○	○			△
Instant Execution	○	○	○			△
Market Execution		○	○			△
Exchange Execution		○	○			△
SL & TP Modification						
Pending Order		○	○	○	○	△
Modify Pending Order				○	○	
Delete Pending Order						

－３．特記；上表、type filling で指定する「Fill Policy」とは

※MT4 では見慣れない概念なので、記載することにしました。

・MQL5(MT5)では、オーダー内容に Fill Policy の指定が追加されています。
Fill Policy とは、オーダーしたロット数（volume）がマーケットに一部しか受け入れられない場合に「オーダー（注文）自体をどうするか」を指定するものです。

◎Fill Policy（フィル・ポリシー；注文ロット数の扱い方針）には以下の「３通り」ある

識別子（type_filling）指定 ENUM_ORDER_TYPE_FILLING type_filling	注文 種類	方針内容
ORDER_FILLING_FOK （Fill or Kill）	成行	・注文したロット数の「全て」が、マーケットに 受け入れられる場合にのみ、注文を実行する。 ・上記以外は、注文をキャンセルする。
ORDER_FILLING_IOC （Immediate or Cancel）	成行	・注文したロット数の「一部」しか、マーケットで 受け入れられない場合、「一部」のみ注文を実行する。 ・残りのロット数は注文をキャンセルする。
ORDER_FILLING_RETURN （Return）	成行	Market orders の場合に適用可能； ・注文したロット数の「一部」しか、マーケットで 受け入れらない場合、先ず「一部」の注文を実行し、 ・残りのロット数は、キャンセルせずに、更に 注文を繰返してゆく。
	指値 逆指値	指値、逆指値の場合； ・注文したロット数の「一部」しか、マーケットで 受け入れらない場合、先ず「一部」の注文を実行し、 ・残りのロット数は、キャンセルせずに、更に 注文を繰返してゆく。

※慣れない概念なので、翻訳等に間違いあれば、ご容赦。

◎「注文実行（Execution）タイプ」と「Filling Policy」の関係

	(Fill or Kill)	(Immediate or Cancel)	(Return)
Request Execution	○	—	—
Instant Execution	○	—	—
Market Execution	○	○	○
Exchange Execution	○	○	○
Pending Order	？（意味あるか？）	？（意味あるか？）	○

※「○印」；指定可能、　「—印」；使用できず、　「？」；良く判らない！です

（２）MqlTradeResult 構造体（３つ目）・・注文（リクエスト）結果をチェックするのに使用

```
struct MqlTradeResult
  {
   uint     retcode;          // Operation return code
   ulong    deal;             // Deal ticket, if it is performed
   ulong    order;            // Order ticket, if it is placed
   double   volume;           // Deal volume, confirmed by broker
   double   price;            // Deal price, confirmed by broker
   double   bid;              // Current Bid price
   double   ask;              // Current Ask price
   string   comment;          // Broker comment to operation
                               (by default it is filled by the operation description)
   uint     request_id;       // Request ID set by the terminal during the dispatch
  };
```

－１．構造体メンバーの内容；

メンバー	内　容
retcode	Return code of a trade server
deal	Deal ticket, if a deal has been performed. It is available for a trade operation of TRADE_ACTION_DEAL type
order	Order ticket, if a ticket has been placed. It is available for a trade operation of TRADE_ACTION_PENDING type
volume	Deal volume, confirmed by broker. It depends on the order filling type
price	Deal price, confirmed by broker. It depends on the *deviation* field of the trade request and/or on the trade operation
bid	The current market Bid price (requote price)
ask	The current market Ask price (requote price)
comment	The broker comment to operation (by default it is filled by the operation description)
request_id	Request ID set by the terminal when sending to the trade server

－２．特記； retcode について

```
struct MqlTradeResult
  {
   uint     retcode;          // Operation return code  ←ここの「メンバー」
    ・・・・・・・・・・  };
```

・retcode メンバー（例えば「retcode==TRADE_RETCODE_PLACED」の意味するところとは？）

Code	Constant (retcode== ○○)	Description (内容)
10004	TRADE_RETCODE_REQUOTE	Requote
10006	TRADE_RETCODE_REJECT	Request rejected
10007	TRADE_RETCODE_CANCEL	Request canceled by trader
10008	**TRADE_RETCODE_PLACED**	**Order placed　・サーバーにオーダーが設定された**
10009	**TRADE_RETCODE_DONE**	**Request completed ・注文（オーダーリクエスト）の実行完了**
10010	TRADE_RETCODE_DONE_PARTIAL	Only part of the request was completed
10011	TRADE_RETCODE_ERROR	Request processing error
10012	TRADE_RETCODE_TIMEOUT	Request canceled by timeout
10013	TRADE_RETCODE_INVALID	Invalid request
10014	TRADE_RETCODE_INVALID_VOLUME	Invalid volume in the request
10015	TRADE_RETCODE_INVALID_PRICE	Invalid price in the request
10016	TRADE_RETCODE_INVALID_STOPS	Invalid stops in the request
10017	TRADE_RETCODE_TRADE_DISABLED	Trade is disabled
10018	TRADE_RETCODE_MARKET_CLOSED	Market is closed
10019	TRADE_RETCODE_NO_MONEY	There is not enough money to complete the request
10020	TRADE_RETCODE_PRICE_CHANGED	Prices changed
10021	TRADE_RETCODE_PRICE_OFF	There are no quotes to process the request
10022	TRADE_RETCODE_INVALID_EXPIRATION	Invalid order expiration date in the request
10023	TRADE_RETCODE_ORDER_CHANGED	Order state changed
10024	TRADE_RETCODE_TOO_MANY_REQUESTS	Too frequent requests
10025	TRADE_RETCODE_NO_CHANGES	No changes in request
10026	TRADE_RETCODE_SERVER_DISABLES_AT	Autotrading disabled by server
10027	TRADE_RETCODE_CLIENT_DISABLES_AT	Autotrading disabled by client terminal
10028	TRADE_RETCODE_LOCKED	Request locked for processing
10029	TRADE_RETCODE_FROZEN	Order or position frozen
10030	TRADE_RETCODE_INVALID_FILL	Invalid order filling type
10031	TRADE_RETCODE_CONNECTION	No connection with the trade server
10032	TRADE_RETCODE_ONLY_REAL	Operation is allowed only for live accounts
10033	TRADE_RETCODE_LIMIT_ORDERS	The number of pending orders has reached the limit
10034	TRADE_RETCODE_LIMIT_VOLUME	The volume of orders and positions for the symbol has reached the limit
10035	TRADE_RETCODE_INVALID_ORDER	Incorrect or prohibited order type
10036	TRADE_RETCODE_POSITION_CLOSED	Position with the specified POSITION_IDENTIFIER has already been closed

（３）MqlTradeTransaction 構造体（４つ目）

※注文（リクエスト）の各処理過程（スタートから終了まで）での内容チェックするのに使用「OrderSendAsync()」と組み合わせて使用されることが多い。
（と言うか、組み合わせると「効果」が発揮される！）

```
struct MqlTradeTransaction
  {
   ulong                     deal;             // Deal ticket
   ulong                     order;            // Order ticket
   string                    symbol;           // Trade symbol name
   ENUM_TRADE_TRANSACTION_TYPE type;           // Trade transaction type
   ENUM_ORDER_TYPE           order_type;       // Order type
   ENUM_ORDER_STATE          order_state;      // Order state
   ENUM_DEAL_TYPE            deal_type;        // Deal type
   ENUM_ORDER_TYPE_TIME      time_type;        // Order type by action period
   datetime                  time_expiration;  // Order expiration time
   double                    price;            // Price
   double                    price_trigger;  // Stop limit order activation price
   double                    price_sl;         // Stop Loss level
   double                    price_tp;         // Take Profit level
   double                    volume;           // Volume in lots
  };
```

－１．構造体メンバーの内容；

メンバー	内　容
deal	Deal ticket.
order	Order ticket.
symbol	The name of the trading symbol, for which transaction is performed.
type	Trade transaction type. The value can be one of ENUM_TRADE_TRANSACTION_TYPE enumeration values.
order_type	Trade order type. The value can be one of ENUM_ORDER_TYPE enumeration values.
order_state	Trade order state. The value can be one of ENUM_ORDER_STATE enumeration values.
deal_type	Deal type. The value can be one of ENUM_DEAL_TYPE enumeration values.
type_time	Order type upon expiration. The value can be one of ENUM_ORDER_TYPE_TIME values.
time_expiration	Pending order expiration term (for orders of ORDER_TIME_SPECIFIED and ORDER_TIME_SPECIFIED_DAY types).
price	Price. Depending on a trade transaction type, it may be a price of an order, a deal or a position.
price_trigger	Stop limit order stop (activation) price (ORDER_TYPE_BUY_STOP_LIMIT and ORDER_TYPE_SELL_STOP_LIMIT).
price_sl	Stop Loss price. Depending on a trade transaction type, it may relate to an order, a deal or a position.
price_tp	Take Profit price. Depending on a trade transaction type, it may relate to an order, a deal or a position.
volume	Volume in lots. Depending on a trade transaction type, it may indicate the current volume of an order, a deal or a position.

－２．特記；TradeTransaction タイプについて

```
struct MqlTradeTransaction
{
   ・・・・・・・
   ENUM_TRADE_TRANSACTION_TYPE type;
   ・・・・・・・
}
```

・ENUM_TRADE_TRANSACTION_TYPE　type メンバーの意味するところ

Trade Transaction Type 注文処理過程	Identifier (type== ○○)	Description (内容)
ステップ１ handling a trade request	TRADE_TRANSACTION_REQUEST	注文（トレード・リクエスト）がトレード・サーバーによって処理されていて、結果も得られ始めた。
ステップ２ changing open oreders	TRADE_TRANSACTION_ORDER_ADD	新規オーダー（new open oredr）を追加
	TRADE_TRANSACTION_ORDER_UPDATE	オーダー（open order）情報の更新。 オーダー・プロセスの進展を含む。 例；オーダーチェック完了、まだブローカは受入れず ⇒［ブローカーが受け入れた］ ⇒［一部、実行された］　など
	TRADE_TRANSACTION_ORDER_DELETE	オーダー（open）リストから削除された。 例；オーダーが実行されることで、 履歴（history）リストに移る場合も含む。
ステップ３ changing orders history	TRADE_TRANSACTION_HISTORY_ADD	履歴（history）にオーダー情報を追加。 オーダー実行やキャンセル結果による。
	TRADE_TRANSACTION_HISTORY_UPDATE	オーダー情報の履歴（history）を更新。
	TRADE_TRANSACTION_HISTORY_DELETE	履歴（history）からオーダー情報を削除。
ステップ４ changing deals history	TRADE_TRANSACTION_DEAL_ADD	ディールが履歴（history）リストに追加された。 オーダー実行の結果等による。
	TRADE_TRANSACTION_DEAL_UPDATE	ディール履歴（history）の更新。 例；　以前、実行されたディールが変更された。 例えばExchange Exexcution で、 ブローカのサーバーから、 外部のトレーディング・システムに委託した ディールが変更された場合など。
	TRADE_TRANSACTION_DEAL_DELETE	履歴（history）からディール結果を削除。 例；　以前、実行されたディールがサーバーから削除。 例えばExchange Exexcution で ブローカーのサーバーから 外部のトレーディング・システムに委託した ディールが削除された場合など。
ステップ５ changing a positions	TRADE_TRANSACTION_POSITION	ポジションの中身が変更された。 例；　ロット数、価格、ストップロス、 テイク・プロフィット等の 内容などが変更された。

５．MQL5 コード例

（１）ダイレクトにトレード結果をフォロー（解析）する

－１．全体コード構成（例）

※スクリプトなら「OnStart()」内で、ＥＡなら「OnChartEvent()等」内で

```
{
   MqlTradeRequest request=(0);
   MqlTradeResult result=(0);

   request.action=TRADE_ACTION_DEAL;
    ・・・・・・・・・・・・
   request.type=ORDER_TYPE_BUY;
    ・・・・・・・・・・・・
    ・・・・・・・・・・・・
   OrderSend(request,result);
    ・・・・・・・・・・・・
   if(result.retcode==〇〇〇) // retcode の状況によって分岐する
   {
      ・・・・・
   }
    ・・・・・・・・・・・・
}
```

－２．コード例（ポイント）

＜OrderSend()＞

（例１）

```
MqlTradeRequest result={0};
・・・・・
bool return=OrederSend(request,result);
if(return)
        { ・・注文用のパラメータが正しいことが確認できた場合の処理・・}
```

（例２）

```
MqlTradeRequest result={0};
・・・・・
OrderSend(request,result);
if(result.retcode==TRADE_RETCODE_PLACED)
        {・・注文がサーバーに届いた場合の処理・・}
```

（例３）

```
MqlTradeRequest result={0};
・・・・・
OrderSend(request,result);
if(result.retcode==TRADE_RETCODE_DONE)
        {・・注文が実行完了して約定した場合の処理・・}
```

＜OrderSendAsync()＞

（例１）

```
MqlTradeRequest result={0};
・・・・・
bool return=OrederSendAsync(request,result);
```

```
        if(return)
                { ・・注文がサーバーに届いた場合の処理？？・・}
        ※本件、チョット「_PLACED」との違いに自信がない
    (例２)
        MqlTradeRequest result={0};
        ・・・・・
        OrderSendAsync(request,result);
        if(result.retcode==TRADE_RETCODE_PLACED)
                {・・注文がサーバーに届いた場合の処理・・}
    (例３)
        MqlTradeRequest result={0};
        ・・・・・
        OrderSendAsync(request,result);
        if(result.retcode==TRADE_RETCODE_DONE)
                {・・注文が実行完了して約定した場合の処理・・}
```

（２）割込発生を利用した解析

－１．全体像

各関数と引数「構造体」の関係；

トレード関数	
bool OredrSend(MqlTradeRequest &request, MqlTradeResult &result);	bool OredrSendAsync(MqlTradeRequest &request, MqlTradeResult &result);
通常のトレード用	高速トレード用

⇒

割込み処理（トレード結果を解析する）	
OnTrade()	OnTradeTransaction(const MqlTradeTransaction &trans, const MqlTradeRequest &request, const MqlTradeResult &result);
[－２．] 参照	[－３．]参照 詳細処理プロセスを捉えることが可能

※「OredrSend」と「OredrSendAsync」は、「OnTrade」と「OnTradeTransaction」の両方共に利用可能で、トレード処理状況を把握することが可能。（たぶん）
ただ、高速トレード「OredrSendAsync」では、「OnTradeTransaction」で処理状況を詳細に把握していく必要があると思われます。

－２．OnTrade()による解析で判ること

※１；OnTrade() 割り込みで、解析可能な情報例（下記の値の変化）

使用する関数 （この値が**変化したとき割込発生**）	内容（概要）
PositionsTotal()	現在、所有（オープン）しているポシション総数
OrdersTotal()	トレード・リクエストの総数 ・ペンディング・オーダーは含まず
HIstoryOrdersTotal()	履歴（ヒストリー）リスト中のオーダー総数
HistoryDealsTotal()	履歴（ヒストリー）リスト中のディール総数

－３．OnTradeTransaction()による解析で判ること

※２；OnTradeTransaction() 割り込みで、解析可能な情報例

・「MqlTradeRequest &result」は、OrderSend()、OrderSendAsync()、OnTradeTransaction ()で利用可能だが、得られる結果は「どの処理過程（プロセス)」中にあるかにより異なる。
（プロセス進展と共に、時々刻々、変わってゆく？！）

・「MqlTradeTransaction &trans」により得られる情報（例）；

構造体メンバー（項目） （この値が**変化したとき割込発生**）	備考
Deal ticket	
Order ticket	
Trade Symbol name	
Trade transaction type	＊A
Order type	
Order state	
Deal type	
Order type by action period	
Order expiration time	
Price	
Stop limit order activation price	
Stop Loss level	
Take Profit level	
Volume	

＊Aの項目；詳細解説は省略（なんとなく判り、ますよね！？）

・Trade transaction type

Identifier（Trade transaction type==○○）
TRADE_TRANSACTION_ORDER_ADD
TRADE_TRANSACTION_ORDER_UPDATE
TRADE_TRANSACTION_ORDER_DELETE
TRADE_TRANSACTION_DEAL_ADD
TRADE_TRANSACTION_DEAL_UPDATE
TRADE_TRANSACTION_DEAL_DELETE
TRADE_TRANSACTION_HISTORY_ADD
TRADE_TRANSACTION_HISTORY_UPDATE
TRADE_TRANSACTION_HISTORY_DELETE
TRADE_TRANSACTION_POSITION ＜要約＞ ディールの実行結果（マーケット上）ではなく、 別の理由でトレードサーバー上でポシション内容が変更された。 ・ロット数、オープン価格、ストップロスとテイクプロフィットのレベル等が変更された。
TRADE_TRANSACTION_REQUEST ＜要約＞ トレード・サーバーによりトレード・リクエストが処理された。 ・そして、処理結果がターミナル側で受信された。

－４．全体コード構成（例）

```
  int OnInit() { 初期設定  return(0);]
//---------------
  void OnDeinit() { 事後処理 ]
//-----リクエスト等の発行元
  void OnTick()
        {
          ・・・・ OrderSend()、やOrderSendAsync(request、result)を記述・・・・
        }
//リクエスト等の発行元;チャート上のオブジェクトをクリックすると、注文が発生する、とか！
  void OnChartEvent(
                const int id,
                const long &lparam,
                const double &dparam,
                const string &sparam    )
        {
          ・・・・ OrderSend()、やOrderSendAsync(request、result)を記述・・・・
        }
//--------------------------------
//トレード処理経過による割込み発生を利用する場合
  void OnTrade()
        {
          //----現在のデータを入手（例）
                int orders=OrdersTotal();
                ・・・・・・・・・・
          //----ヒストリー・データを入手（例）
                bool selected=HistorySelect(start,end);
                int history_orders=HistoryOrdersTotal();
                ・・・・・・・・・・
        }
//--------------------------------
//トレード処理経過による割込み発生を利用する場合
//MqlTradeTrabsaction &trans を利用した詳細情報の取得
  void OnTradeTransaction(
                const MqlTradeTrabsaction &trans,
                const MqlTradeRequest &request,
                const MqlTradeResult &result     )
        {
           ・・・・
           //  例１
           ENUM_TRADE_TRANSACTION_TYPE type=trans.type;
           if(type==TRADE_TRANSACTION_REQUEST)
                {
                    Print("Order ticket: "+(string)trans.order+"¥r¥n";
                    ・・・・・
                }
           ・・・・
            //  例２
            if(HistoryDealGetInteger(trans.deal,DEAL_ENTRY)==DEAL_ENTRY_IN)
            {
                if(trans.volume != 0 && trans.type != TRADE_TRANSACTION_HISTORY_UPDATE)
                {
                  ・・・・・・・・・・
                }
            }
        }
```

（３）詳細コード（例）

＜OrderSendAsync()と Experts.log の関係＞

log　出力コード	Experts.log 出力の該当部
void OnChartEvent(const int id, const long &lparam, const double &dparam, const string &sparam) { //--- handling CHARTEVENT_CLICK event ("Clicking the chart") if(id==CHARTEVENT_OBJECT_CLICK) { Print("=> ",__FUNCTION__,": sparam = ",sparam); //--- minimum volume for a deal double volume_min=SymbolInfoDouble(_Symbol,SYMBOL_VOLUME_MIN); //--- if "Buy" button is pressed, then buy if(sparam=="Buy") { PrintFormat("Buy %s %G lot",_Symbol,volume_min);//① BuyAsync(volume_min); //--- unpress the button ObjectSetInteger(0,"Buy",OBJPROP_STATE,false); } //--- if "Sell" button is pressed, then sell if(sparam=="Sell") { PrintFormat("Sell %s %G lot",_Symbol,volume_min);//② SellAsync(volume_min); //--- unpress the button ObjectSetInteger(0,"Sell",OBJPROP_STATE,false); } ChartRedraw(); } //--- }	=> OnChartEvent: sparam = Sell *if(sparam=="Buy")* //① Buy EURUSD 0.01 lot *if(sparam=="Sell")* //② Sell EURUSD 0.01 lot
void BuyAsync(double volume) { //--- prepare the request MqlTradeRequest req={0}; req.action　=TRADE_ACTION_DEAL; req.symbol　=_Symbol; req.magic　=MagicNumber; req.volume　=0.1; req.type　=ORDER_TYPE_BUY; req.price　=SymbolInfoDouble(req.symbol,SYMBOL_ASK); req.deviation　=10; req.comment　="Buy using OrderSendAsync()"; MqlTradeResult　res={0}; if(!OrderSendAsync(req,res)) { Print(__FUNCTION__,": error ",GetLastError(), ", retcode = ",res.retcode); } //--- }	
void SellAsync(double volume) { //--- prepare the request MqlTradeRequest req={0}; req.action　=TRADE_ACTION_DEAL; req.symbol　=_Symbol; req.magic　=MagicNumber; req.volume　=0.1; req.type　=ORDER_TYPE_SELL; req.price　=SymbolInfoDouble(req.symbol,SYMBOL_BID); req.deviation　=10; req.comment　="Sell using OrderSendAsync()"; MqlTradeResult　res={0}; if(!OrderSendAsync(req,res)) { Print(__FUNCTION__,": error ",GetLastError(), ", retcode = ",res.retcode); } //--- }	

```mql5
void OnTradeTransaction(const MqlTradeTransaction &trans,
                        const MqlTradeRequest &request,
                        const MqlTradeResult &result)
  {
//--- heading named after trading event's handler function
   Print("=> ",__FUNCTION__," at ",
                  TimeToString(TimeCurrent(),TIME_SECONDS));
//--- receive transaction type as enumeration value
   ENUM_TRADE_TRANSACTION_TYPE type=trans.type;
//--- if transaction is a result of request handling
   if(type==TRADE_TRANSACTION_REQUEST)
     {
      //--- display transaction name
      Print(EnumToString(type));//③
      //--- then display the string description
                    of the handled request
      Print("------------RequestDescription¥r¥n",
          RequestDescription(request,DescriptionModeFull));//④
      //--- and show description of the request result
      Print("------------ ResultDescription¥r¥n",
          TradeResultDescription(result,DescriptionModeFull));//⑤
     }
   else // display full description of the transaction
                   for transactions of another type
     {
      Print("------------ TransactionDescription¥r¥n",
          TransactionDescription(trans,DescriptionModeFull));//⑥
     }
//---
  }
```

＜割込み発生＞

```
=> OnTradeTransaction at 09:52:53
```

if(type==TRADE_TRANSACTION_REQUEST)

③ TRADE_TRANSACTION_REQUEST

④ ------------ RequestDescription
以下（４）参照

⑤ ------------ ResultDescription
以下（５）参照

else

⑥ ------------ TransactionDescription
以下（６）参照

```mql5
string RequestDescription(const MqlTradeRequest &request,
                          const bool detailed=true)
  {
//--- prepare a string for returning from the function
   string desc=EnumToString(request.action)+"¥r¥n";
//--- add all available data in detailed mode
   if(detailed)
     {
      desc+="Symbol: "+request.symbol+"¥r¥n";
      desc+="Magic Number: "
                        +StringFormat("%d",request.magic)+"¥r¥n";
      desc+="Order ticket: "+(string)request.order+"¥r¥n";
      desc+="Order type: "+EnumToString(request.type)+"¥r¥n";
      desc+="Order filling: "
                        +EnumToString(request.type_filling)+"¥r¥n";
      desc+="Order time type: "
                        +EnumToString(request.type_time)+"¥r¥n";
      desc+="Order expiration: "
                        +TimeToString(request.expiration)+"¥r¥n";
      desc+="Price: "+StringFormat("%G",request.price)+"¥r¥n";
      desc+="Deviation points: "
                     +StringFormat("%G",request.deviation)+"¥r¥n";
      desc+="Stop Loss: "+StringFormat("%G",request.sl)+"¥r¥n";
      desc+="Take Profit: "+StringFormat("%G",request.tp)+"¥r¥n";
      desc+="Stop Limit: "
                     +StringFormat("%G",request.stoplimit)+"¥r¥n";
      desc+="Volume: "+StringFormat("%G",request.volume)+"¥r¥n";
      desc+="Comment: "+request.comment+"¥r¥n";
     }
//--- return the received string
   return desc;
  }
```

（４）

```
TRADE_ACTION_DEAL

if(detailed)

Symbol: EURUSD
Magic Number: 1234567
Order ticket: 16361998
Order type: ORDER_TYPE_SELL
Order filling: ORDER_FILLING_FOK
Order time type: ORDER_TIME_GTC
Order expiration: 1970.01.01 00:00
Price: 1.29313
Deviation points: 10
Stop Loss: 0
Stop Limit: 0
Volume: 0.1
Take Profit: 0
Stop Limit: 0
Comment: Sell using OrderSendAsync()
```

```mql5
string TradeResultDescription(const MqlTradeResult &result,
                              const bool detailed=true)
  {
//--- prepare the string for returning from the function
   string desc="Retcode "+(string)result.retcode+"¥r¥n";
//--- add all available data in detailed mode
   if(detailed)
     {
      desc+="Request ID: "
                  +StringFormat("%d",result.request_id)+"¥r¥n";
      desc+="Order ticket: "+(string)result.order+"¥r¥n";
      desc+="Deal ticket: "+(string)result.deal+"¥r¥n";
      desc+="Volume: "+StringFormat("%G",result.volume)+"¥r¥n";
      desc+="Price: "+StringFormat("%G",result.price)+"¥r¥n";
      desc+="Ask: "+StringFormat("%G",result.ask)+"¥r¥n";
```

（５）

```
Retcode 10009  //==TRADE_RETCODE_DONE

 if(detailed)
Request ID: 2
Order ticket: 16361998
Deal ticket: 15048668
Volume: 0.1
Price: 1.29313
Ask: 1.29319
```

```
      desc+="Bid: "+StringFormat("%G",result.bid)+"¥r¥n";
      desc+="Comment: "+result.comment+"¥r¥n";
     }
//--- return the received string
   return desc;
  }
```

```
Bid: 1.29313
Comment:
```

```
string TransactionDescription(
        const MqlTradeTransaction &trans,const bool detailed=true)
  {
//--- prepare a string for returning from the function
   string desc=EnumToString(trans.type)+"¥r¥n";
//--- all possible data is added in detailed mode
   if(detailed)
     {
      desc+="Symbol: "+trans.symbol+"¥r¥n";
      desc+="Deal ticket: "+(string)trans.deal+"¥r¥n";
      desc+="Deal type: "+EnumToString(trans.deal_type)+"¥r¥n";
      desc+="Order ticket: "+(string)trans.order+"¥r¥n";
      desc+="Order type: "+EnumToString(trans.order_type)+"¥r¥n";
      desc+="Order state: "+EnumToString(trans.order_state)+"¥r¥n";
      desc+="Order time type: "+EnumToString(trans.time_type)+"¥r¥n";
      desc+="Order expiration: "
                 +TimeToString(trans.time_expiration)+"¥r¥n";
      desc+="Price: "+StringFormat("%G",trans.price)+"¥r¥n";
      desc+="Price trigger: "
                 +StringFormat("%G",trans.price_trigger)+"¥r¥n";
      desc+="Stop Loss: "+StringFormat("%G",trans.price_sl)+"¥r¥n";
      desc+="Take Profit: "+StringFormat("%G",trans.price_tp)+"¥r¥n";
      desc+="Volume: "+StringFormat("%G",trans.volume)+"¥r¥n";
     }
//--- return a received string
   return desc;
  }
```

（６）例

```
TRADE_TRANSACTION_ORDER_ADD //プロセス
Symbol: EURUSD
Deal ticket: 0
Deal type: DEAL_TYPE_BUY
Volume: 0.1
Order type: ORDER_TYPE_SELL
Order state: ORDER_STATE_STARTED
Order time type: ORDER_TIME_GTC
Order expiration: 1970.01.01 00:00
Price: 1.29313
Price trigger: 0
Stop Loss: 0
Take Profit: 0
Order ticket: 16361998
↓
TRADE_TRANSACTION_ORDER_DELETE //プロセス
・・・・・・・・・・・・・・・・・・・・・・
↓
TRADE_TRANSACTION_HISTORY_ADD //プロセス
・・・・・・・・・・・・・・・・・・・・・・
↓
TRADE_TRANSACTION_DEAL_ADD //プロセス
・・・・・・・・・・・・・・・・・・・・・・
```

```
void OnTrade()
  {
//--- static members for storing trading account status
   static int prev_positions=0,prev_orders=0,prev_deals=0,
              prev_history_orders=0;
//--- request trading history
   bool update=HistorySelect(history_start,TimeCurrent());
   PrintFormat("HistorySelect(%s , %s) = %s",
                    TimeToString(history_start),
    TimeToString(TimeCurrent()),(string)update);   //⑦
//--- heading named after trading event's handler function
   Print("=> ",__FUNCTION__," at ",

TimeToString(TimeCurrent(),TIME_SECONDS));
//--- display handler's name and
      the number of orders at the moment of handling
   int curr_positions=PositionsTotal();
   int curr_orders=OrdersTotal();
   int curr_deals=HistoryOrdersTotal();
   int curr_history_orders=HistoryDealsTotal();
//--- display the number of orders,
           positions, deals, as well as changes in parentheses
   PrintFormat("PositionsTotal() = %d (%+d)",
          curr_positions,(curr_positions-prev_positions));  //⑧
   PrintFormat("OrdersTotal() = %d (%+d)",
                curr_orders,curr_orders-prev_orders);        //⑨
   PrintFormat("HistoryOrdersTotal() = %d (%+d)",
                curr_deals,curr_deals-prev_deals);           //⑩
   PrintFormat("HistoryDealsTotal() = %d (%+d)",curr_history_orders,
                curr_history_orders-prev_history_orders);    //⑪
//--- insert a string break to view the log more conveniently
   Print("");
//--- save the account status
   prev_positions=curr_positions;
   prev_orders=curr_orders;
   prev_deals=curr_deals;
   prev_history_orders=curr_history_orders;
//---
  }
```

＜割込発生＞

（例）

```
//⑦
HistorySelect( 09:34 , 09:52) = true

=> OnTrade at 09:52:53
```

```
//⑧～⑪

PositionsTotal() = 1 (+1)
OrdersTotal() = 0 (+0)
HistoryOrdersTotal() = 2 (+2)
HistoryDealsTotal() = 2 (+2)
```

以　上