Use of MetaTrader 4 Manager API

Permalink: https://support.metaquotes.net/en/articles/32

22 November 2005

MetaTrader Manager API represents a library in form of a DLL file containing the complete set of administrator and manager commands to access to MetaTrader Server. Those who use the Manager API can write their own administrator or manager utilities, or even their own manager terminal.

Manager API library and examples of using thereof are distributed together with MetaTrader Administrator and located in \api directory of the administrator. New versions of the Manager API are included into the LiveUpdate system and automatically downloaded together with other components thereof. Some examples with source codes are distributed together with the Manager API: ManagerAPIAdmin - use of the administrator functions; ManagerAPISample - use of the manager functions, pumping, dealing; ManagerAPITrade - an example of the manager trading transaction; DelphiSample - an example of using API in Delphi environment; DelphiPumping - an example of using the pumping in Delphi environment.

Exported functions and manager interface

API as a whole is described in a header file '\api\MT4ManagerAPI.h', 'ManagerAPI.pas' file applies for examples on Delphi. Manager API library exports two simple functions:

int MtManVersion() — returns the version and the build of the API library, the high-order word contains API version, the low-order word does the build. To mark out the version and the build from the returned value, macros named HIWORD() and LOWORD() can be used.

int MtManCreate(int version,CManagerInterface **man) — creates and returns a manager interface, it returns zero when there is an error in interface creation, otherwise, it returns a nonzero value.

'MT4ManagerAPI.h' file includes manager interface factory **CManagerFactory** that loads 'mtmanapi.dll' library automatically, allows to initialize the WinSocks library, and gives wrappers over exported functions named MtManVersion() and MtManCreate().

CManagerInterface class represents a manager interface: the set of administrator and manager functions to access to MetaTrader Server. Every manager interface can be in three modes:

- Direct connection to the server
  can be used to configure the server by administrator commands, work with backups, access directly to the server databases, change symbol settings, send mails, news, etc.
- Pumping
  connecting in the mode of data spooling, manager interface getting changes of databases from the server in a saving way and sending notifications about the updated data to the user program.
- Dealing
  connecting in the mode of processing the clients' trading requests, manager interface sending notifications about the income of new trading requests to the user program and sending dealer's responses to the server.

After the manager interface has been created, the program can call the **WorkingDirectory()** function to set API working directory where symbol settings, received mails and API journal will be stored. If an error occurs the **ErrorDescription()** function returns a text description of the error. After the work with the manager interface has been finished, it is necessary to call the **Release()** function to delete the previously created manager interface.

> **Important:** Many functions of manager interface, such as CfgRequestAccess() or UsersRequest(), return a pointer to a data array which, after being used, must be freed through MemFree() function.

Connection and authorization

Connection to the server is performed with **Connect()** function that accepts the server address formatted as 'server:port'. If there is no port specified, port 443 will be used by default. **Disconnect()** function allows to disconnect from the server. **IsConnected()** function allows to check the connection status and returns a nonzero value if manager interface is connected to the server, otherwise, it returns zero.

Authorization on the server is performed through **Login()** function. Using the Manager API, it is possible to connect only accounts from the 'manager' group for which the rights were registered in MetaTrader Administrator in the Managers line.

When applying Advanced Security to the 'manager' group, authorization with RSA keys is required. At that, calling the

Login() function will return one of the following values:

- **RET_GENERATE_KEY** — at the first connection of the account, the server requires a public key to be sent to it through **KeysSend()** function. RSA keys can be created with MetaTrader Administrator or MetaTrader Manager. After the key has been sent it is necessary to disconnect and then reconnect and be authorized on the server.
- **RET_SECURITY_SESSION** — the expanded authorization with RSA keys is required. To do so, it is necessary to call **LoginSecured()** function having sent the path to the RSA-keys file as a parameter.

After having been authorized on the server, one can change one's own manager account password through **PasswordChange()** function, get one's own rights settings with **ManagerRights()** function, request server time with **ServerTime()** function, or check connection to the server with **Ping()** function.

Administrator commands

Manager API contains a number of functions that, to be successfully executed, need manager account to have administrator rights.

Manager interface includes four functions sending commands to manage the server operation:

- **SrvRestart()** — restart MetaTrader Server;
- **SrvChartsSync()** — synchronize history data;
- **SrvLiveUpdateStart()** — start LiveUpdate;
- **SrvFeedsRestart()** — restart data feeds.

Server functionality can be expanded by adding commands to be implemented by the server plugins. When the manager interface function of **ExternalCommand()** has been called the server gives the received command to plugins that export the **MtSrvManagerProtocol()** function.

Manager interface includes a number of functions used to change server settings:

- **CfgRequest*()** — request for server settings;
- **CfgUpdate*()** — change a certain record of the server configuration;
- **CfgDelete*()** — delete a certain record of the server configuration;
- **CfgShift*()** — shift a certain record of the server configuration.

All available on the server data feeds can be requested for with the **SrvFeeders()** function, and the **SrvFeederLog()** function allows to request for the logs of the data feed configured by name.

Manager interface includes functions to manage the server history databases:

- **ChartRequest()** — request for history data by symbol and timeframe from the certain moment of time;
- **ChartAdd()** — add bars to the history database;
- **ChartUpdate()** — update bars in the history database;
- **ChartDelete()** — delete bars from the history database.

The server performance database is accessed with **PerformanceRequest()** function from a certain time. User program operating with Manager API must support the local performance database by itself after the fisrt request for the server performance and it must only request for missing data only, if necessary. Regular requests for the entire database of the server preformance are forbidden.

Manager interface includes administrator functions of access to the current databases of the server:

- **AdmUsersRequest()** — request for accounts from the current database; the list of groups or accounts separated by commas can be specified as the request string;
- **AdmTradesRequest()** — request for orders of the current database; list of groups, accounts, or orders, separated by commas, can be specified as the request string;
- **AdmTradesDelete()** — deleting of orders from the current database;
- **AdmTradeRecordModify()** — modifying of an order in the current database;
- **AdmBalanceCheck()** — checking of balance of the account list;
- **AdmBalanceFix()** — correcting of balance of the account list according to the trade history.

Manager interface includes functions to work with database backups of accounts and orders:

- **BackupInfoUsers()** — request for the file list of backups of the account database;

- **BackupInfoOrders()** — request for the file list of backups of the order database;
- **BackupRequestUsers()** — request for accounts from a specific file of the backup; a list of groups or accounts separated by commas can be specified as the request string;
- **BackupRequestOrders()** — request for orders from a specific file of the backup; a list of groups or accounts separated by commas can be specified as the request string;
- **BackupRestoreUsers()** — restoring of accounts from a backup;
- **BackupRestoreOrders()** — restoring of orders from a backup.

Important: After accounts or orders have been restored from backups it is necessary to correct balances of the restored accounts by calling the AdmBalanceFix() function.

Important: Many administrator functions, such as CfgRequest*(), SrvFeeders(), ChartRequest(), BackupInfo*(), BackupRequest*(), BackupRestoreOrders(), AdmUsersRequest(), AdmTradesRequest(), return pointer to the data array that must be freed after use with the MemFree() function.

Symbols

The list of available symbols can be requested from the server with the **SymbolsRefresh()** function. At the first call, the full symbol list is downloaded from the server that is saved in the working directory of the Manager API in the 'symbols.raw' file. At the posterior calls, symbols are only downloaded if there are changes in symbol settings compared to the saved ones. Settings of the requested symbols or of a specific symbol can be obtained with the following functions: **SymbolsGetAll()** and **SymbolGet()**. Settings of securities groups can be obtained with **SymbolsGroupsGet()** function.

Manager interface supports, along with the full symbol list, the list of selected symbols; in the pumping mode, the manager interface receives quotes updates only for symbols listed as selected symbols. A symbol can be added to the list of selected symbols with **SymbolAdd()** function, the **SymbolHide()** function will remove the symbol from the list of selected symbols. Short description of a selected symbol can be obtained with the **SymbolInfoGet()** function.

If the manager account has the right for 'market watch' it is possible to change spread, execution mode, limit & stop level, and the background color of the given symbol with the **SymbolChange()** function. Using the **SymbolSendTick()** function, it is possible to throw in a quote in the quotes flow from data feeds.

Direct access to the server databases

Manager interface includes a number of functions of direct request and change of the server databases.

The **GroupsRequest()** function of manager interface allows to request a list of available groups of accounts. The list of all accounts can be requested by the **UsersRequest()** function, the list of certain accounts can be requested by the **UserRecordsRequest()** function, the list of the connected clients can be requested by the **OnlineRequest()** function. The **UserRecordNew()** function allows to select a new account, the **UserRecordUpate()** function allows to change these accounts. The **UserGroupOp()** function allows to conduct a group operation over the list of accounts: remove accounts, permit accounts, lock accounts, change leverage or a group of the account list. The **UserPasswordCheck()** function allows to check the account password. The **UserPasswordSet()** function is intended for changing of the main or investor's password of the account and, if Advanced Security mode is enabled, allows to reset the public key on the server, at the next connection, the server will request a new public key from the client.

The list of all orders can be requested with the **TradesRequest()** function, the list of certain orders can be requested with the **TradeRecordsRequest()** function, and the **TradesUserHistory()** function can help to request the trading history for an account.

Manager interface allows to open an order, to close an order, or to modify an open order with the **TradeTransaction()** function. After a new order has been opened or a balance operation has been performed, the new order ticket will be returned into the TradeTransInfo::order field. If necessary, the **TradeCheckStops()** function allows to check the Stop Loss and Take Profit levels of an order, as well as the pending order open price transferred to TradeTransInfo and price transferred as the second parameter of satisfaction the Limit & Stop level specified in Administrator for each symbol (ConSymbol::stops_level). The pending order expiration time will also be checked for being set at no nearer than 10 minutes from the actual time.

Manager interface allows to request for the list of closed positions at which reports can be generated for a certain set of accounts with the **ReportsRequest()** function. Daily reports generated on the server for the list of accounts can be requested with the **DailyReportsRequest()** function. When reports are requested, the report period (from-to fields of ReportGroupRequest and DailyGroupRequest) must be conformed the beginning and end of the server trading day (ConCommon::endhour,ConCommon::endminute). For example, when requesting for the current trading day reports from the server where the end of the trading day is set for 23:59, one can use the following code:

```
ReportGroupRequest req={"some_group"};
req.from=(time(NULL)/86400)*86400;
req.to  =req.from+86400;
...
```

Manager interface allows to request last mails of internal mail system with MailsRequest() function, send a message by the internal mailing system with the **MailSend()** function, throw in a news into the news flow with the **NewsSend()** function, configuration of the server plugin can be changed with the **PluginUpdate()** function. The **JournalRequest()** function allows to request for the server log for a certain period of time.

Pumping

Pumping is a saving and quick mode of uploading data from the server. When transferring to pumping mode, manager interface requests the server for symbol settings, groups, account databases, orders, and trading requests, after that the server sends only updated data to the connected manager. After having connected to the server, the manager interface is transferred to the pumping mode with the **PumpingSwitch()** function. The pointer to the callback function to be used by the manager interface to notify about the data updating, or window handle and identifier of the user message to which the notification about the data updating will be sent, must be passed as a parameter of this function.

After the **PumpingSwitch()** has been called, the manager interface will send the following codes as a parameter of the callback function or as WPARAM of the user message:

- **PUMP_START_PUMPING** — manager interface has successfully changed for the pumping mode;
- **PUMP_UPDATE_SYMBOLS** — symbol settings have been updated, the updated symbols can be obtained with functions named **SymbolsGetAll()**, **SymbolGet()**, or **SymbolInfoGet()**;
- **PUMP_UPDATE_GROUPS** — group settings have been updated, the updated group settings can be obtained with functions named **GroupsGet()** or **GroupRecordGet();**
- **PUMP_UPDATE_USERS** — the account list has been updated, the updated account list can be obtained with functions named **UsersGet()** or **UserRecordGet()**;
- **PUMP_UPDATE_ONLINE** — the list of connected clients has been updated, the updated list can be obtained with functions named **OnlineGet()** or **OnlineRecordGet()**;
- **PUMP_UPDATE_BIDASK** — a new quote has income, updated quotes can be obtained with **TickInfoLast()** function, the updated prices can be obtained with one or several calls of the **SymbolInfoUpdated()** function;
- **PUMP_UPDATE_NEWS** — a news has income, the headings of the income news can be obtained with functions named **NewsGet()**, **NewsTotal()**, and **NewsTopicGet()**; after the news has income it will be possible to request for its body with the **NewsBodyRequest()** function;
- **PUMP_UPDATE_NEWS_BODY** — the news body has income, it can be obtained with **NewsBodyGet()** function;
- **PUMP_UPDATE_MAIL** — the new message has income that was saved on the disk in 'mailbox\' folder of the working directory of Manager API; the file name and the size of the new message can be obtained with the **MailLast()** function;
- **PUMP_UPDATE_TRADES** — the orders list has been updated, the updated list of orders can be obtained with functions named **TradesGet()**, **TradesGetBySymbol()**, **TradesGetByLogin()**, **TradesGetByMarket()**, and **TradeRecordGet()**. The updated trade summary for symbols and for security groups can be obtained with functions named **SummaryGetAll()**, **SummaryGet()**, **SummaryGetByCount()**, and **SummaryGetByType()**. The updated company's exposure for currencies can be obtained with functions named **ExposureGet()** and **ExposureValueGet()**;
- **PUMP_UPDATE_REQUESTS** — the list of trade requests has been updated, the updated list can be obtained with functions named **RequestsGet()** and **RequestInfoGet()**;
- **PUMP_UPDATE_PLUGINS** — the list of server plugins has been updated, the updated list can be obtained with functions named **PluginsGet()** and **PluginParamGet()**;
- **PUMP_UPDATE_ACTIVATION** — the list of orders having triggered Stop Loss or Take Profit level, pending orders to be activated and the most unprofitable orders for accounts in the stop out mode, the corresponding activation flags being set in the 'activation' field of the updated orders logs: **ACTIVATION_SL**, **ACTIVATION_TP**, **ACTIVATION_PENDING**, **ACTIVATION_STOPOUT**, or, if the price has rolled back from the activation level: **ACTIVATION_SL_ROLLBACK**, **ACTIVATION_TP_ROLLBACK**, **ACTIVATION_PENDING_ROLLBACK**; if the price has rolled back the activation flag can be cleared with the **TradeClearRollback()** function;
- **PUMP_UPDATE_MARGINCALL** — the list of accounts in margin call mode has been updated, the updated list of margin requirements of accounts can be obtained with functions named **MarginsGet()** and **MarginLevelGet()**;
- **PUMP_STOP_PUMPING** — the pumping mode has been stopped.

Extended pumping

Besides standard pumping mode, in which only notifications about data change come, there is a special mode with replaying of coming transactions. In this mode there is a possibility to receive information about changes in client and order records, as well as changes in group and symbol settings.

For switching to the extended pumping mode, the **PumpingSwitchEx** method is used:

```
int __stdcall PumpingSwitchEx(MTAPI_NOTIFY_FUNC_EX pfnFunc,const int flags,void *param)
```

This method gets the following parameters:

- **pfnFunc** - pointer to the callback function
- **flags** - bit flags of pumping operation:
  - **CLIENT_FLAGS_HIDETICKS** - do not receive ticks
  - **CLIENT_FLAGS_HIDENEWS** - do not receive news
  - **CLIENT_FLAGS_HIDEMAIL** - do not receive emails
  - **CLIENT_FLAGS_SENDFULLNEWS** - receive news together with news body
  - **CLIENT_FLAGS_HIDEONLINE** - do not receive information about clients online
  - **CLIENT_FLAGS_HIDEUSERS** - do not receive list of clients
- **param** - any parameter that must be passed to a callback function

After switching to the extended pumping mode, the callback function is called:

```
void __stdcall NotifyCallBack(int code,int type,void* data,void *param)
```

As parameters, the callback function receives:

- **code** - type of change
- **type** - type of transaction: TRANS_ADD,TRANS_DELETE,TRANS_UPDATE,TRANS_CHANGE_GRP
- **data** - pointer to updated data: type of this pointer depends on the type of change
- **param** - pointer that has been passed as a parameter to the PumpingSwitchEx function

Below are correspondences between **code data** pointer type:

- **PUMP_UPDATE_USERS** - UserRecord structure
- **PUMP_UPDATE_ONLINE** - client's login in the int form
- **PUMP_UPDATE_TRADES** - TradeRecord structure
- **PUMP_UPDATE_NEWS** - NewsTopic structure
- **PUMP_UPDATE_MAIL** - MailBox structure
- **PUMP_UPDATE_SYMBOLS** - ConSymbol structure
- **PUMP_UPDATE_GROUPS** - ConGroup structure
- **PUMP_UPDATE_REQUESTS** - RequestInfo structure

Below is an example of notification receiving about changes in trade records:

```
if(manager->PumpingSwitchEx(PumpingNotify,0,NULL)!=RET_OK)
   printf("pumping switch error");
```

```
void __stdcall PumpingNotify(int code,int type,void *data,void *param)
  {
//--- checks
   if(code==PUMP_UPDATE_TRADES && data!=NULL)
     {
      TradeRecord *trade=(TradeRecord*)data;
      //---
      switch(type)
```

```
         {
          case TRANS_ADD:
             printf("#%d added",trade->order);
             break;
          case TRANS_DELETE:
             printf("#%d closed",trade->order);
             break;
          case TRANS_UPDATE:
             if(trade->login==0) printf("#%d deleted",trade->order);
             else                printf("#%d updated",trade->order);
             break;
         }
      }
   }
```

Processing of stop loss, take profit, and pending orders

In the pumping mode, when a Stop Loss, a Take Profit, or a pending order triggers, manager interface sends notification of **PUMP_UPDATE_ACTIVATION** to the client program. After the **TradesGet()** has been called the corresponding activation lines will be flagged in the TradeRecord::activation field of updated records of orders: **ACTIVATION_SL**, **ACTIVATION_TP**, or **ACTIVATION_PENDING**. If the price has rolled back from the order activation level, **ACTIVATION_SL_ROLLBACK**, **ACTIVATION_TP_ROLLBACK**, or **ACTIVATION_PENDING_ROLLBACK** will be flagged. An exceeded Stop Loss or Take Profit order can be triggerred or a pending order can be opened with the **TradeTransaction()** function. If, for some reason, there is no need to trigger the order after the price has rolled back from the order activation level, the **TradeClearRollback()** function allows to unflag **ACTIVATION_SL_ROLLBACK**, **ACTIVATION_TP_ROLLBACK**, and **ACTIVATION_PENDING_ROLLBACK** in the orders database of the manager interface.

Margin requirements and processing of Stop Outs

When changing for the pumping mode, the manager interface requests in addition group settings and full accounts and orders databases from the server, and the databases will be economically updated when updates income from the server. Manager interface calculates margin requirements for accounts on basis of the data requested. For better efficiency, margin requirements are calculated only for accounts that have open positions. For accounts that have not had any open positions for the connection period, **MarginLevelGet()** returns RET_ERROR, and margin=0, equity=margin_free=UserRecord::balance+UserRecord::credit at that. If one or more accounts enter or leave the Margin Call status, manager interface sends **PUMP_UPDATE_MARGINCALL** notification to the client program in the pumping mode. Since margin requirements (MarginLevel::margin) change only at opening or closing of positions, **MarginsGet()** should be better called after the **PUMP_UPDATE_TRADES** notification has income. Normally, there is a need to watch only those accounts that are in Margin Call or Stop Out status, and **MarginsGet()** can be called even less frequently: at the **PUMP_UPDATE_MARGINCALL** notification incoming, then standard margin requirements should be removed from the returned array (those for which MarginLevel::level_type==MARGINLEVEL_OK) and, after the **PUMP_UPDATE_TRADES** notification has income, only the remained margin requirements should be updated on logins with **MarginLevelGet()** calls. The database of margin requirements is indexed by logins, so a few **MarginLevelGet()** calls cost cheaper than a **MarginsGet()** call requiring to copy a large array of data for all the accounts. If there is a need to watch free margin (MarginLevel::margin_free) and margin level (MarginLevel::margin_level), **MarginLevelGet()** may also be called for each tick (**PUMP_UPDATE_BIDASK**).

If one or more accounts enter or leave the Stop Out status, the manager interface finds the most unprofitable orders for those accounts automatically and flags **ACTIVATION_STOPOUT** or **ACTIVATION_STOPOUT_ROLLBACK** in the TradeRecord::activation field. After that, the manager interface sends the **PUMP_UPDATE_ACTIVATION** notification to the client program. The latter, after having received this notification, can request for all orders with the **TradesGet()** function and then find and close orders with flagged **ACTIVATION_STOPOUT** or **ACTIVATION_STOPOUT_ROLLBACK** in the received order. If there is no need to close the order for some reason, the **TradeClearRollback()** function allows to unflag **ACTIVATION_STOPOUT_ROLLBACK** in the orders database of the manager interface.

If there is a need to process Stop Out by a method different from closing the most unprofitable position, then it is necessary to request the list of margin levels at the event of **PUMP_UPDATE_MARGINCALL**, find margin levels in the MARGINLEVEL_STOPOUT status in the obtained list, request the list of positions for the obtained logins with the **TradesGetByLogin()**, and find and close the desired position independently, according to the rules of Stop Out processing.

Calculation of profit

The pumping mode includes the option of profit calculation for an order. The special method **TradeCalcProfit()** is used

for this purpose. Here is the example of profit calculation code for BUY 1.00 EURUSD order with the open price = 1.0000 and close price = 1.2000 for the client 5555:

```
TradeRecord trade={0};
COPY_STR(trade.symbol,"EURUSD");
trade.login      =5555;
trade.volume     =100;
trade.cmd        =OP_BUY;
trade.open_price =1.0000;
trade.close_price=1.2000;
if(manager->TradeCalcProfit(™)!=FALSE) printf("%.2lf\n",trade.profit);
```

Dealing

Dealing is the mode of manager interface operation where it receives clients' trading requests from the server and sends the dealer's responses to the server. After connection to the server manager interface is translated to the dealing mode with the **DealerSwitch()** function. Pointer to the callback function to be used by the manager interface to notify about receiving of new trading requests from the server, or window handle and identifier of the user message to which the notification about new trading requests from the server will be sent, must be passed as a parameter of this function.

After the **DealerSwitch()** has been called the manager interface will send the following codes as a parameter of the callback function or as WPARAM of the user message:

- **DEAL_START_DEALING** — manager interface has successfully transferred to the dealing mode;
- **DEAL_REQUEST_NEW** — a new trading request has income from the server;
- **DEAL_STOP_DEALING** — the dealing mode has been stopped.

After it has received the **DEAL_REQUEST_NEW** notification the user program must call the **DealerRequestGet()** function to receive a new trading request, then the dealer can confirm (or requote) the request with **DealerSend()**, reject the request with **DealerReject()**, or return the request to the queue of requests on the server with **DealerReset()** for other dealer to take this request for processing. To requote with **DealerSend()**, it is necessary to set new requoting prices in the RequestInfo::prices[2] fields and send TRUE as the requote parameter. The correct_prices flagged in the **DealerSend()** function defines whether quotes must be thrown in when responding to the request and it corresponds with the Throw In Prices option in the Manager Terminal.
