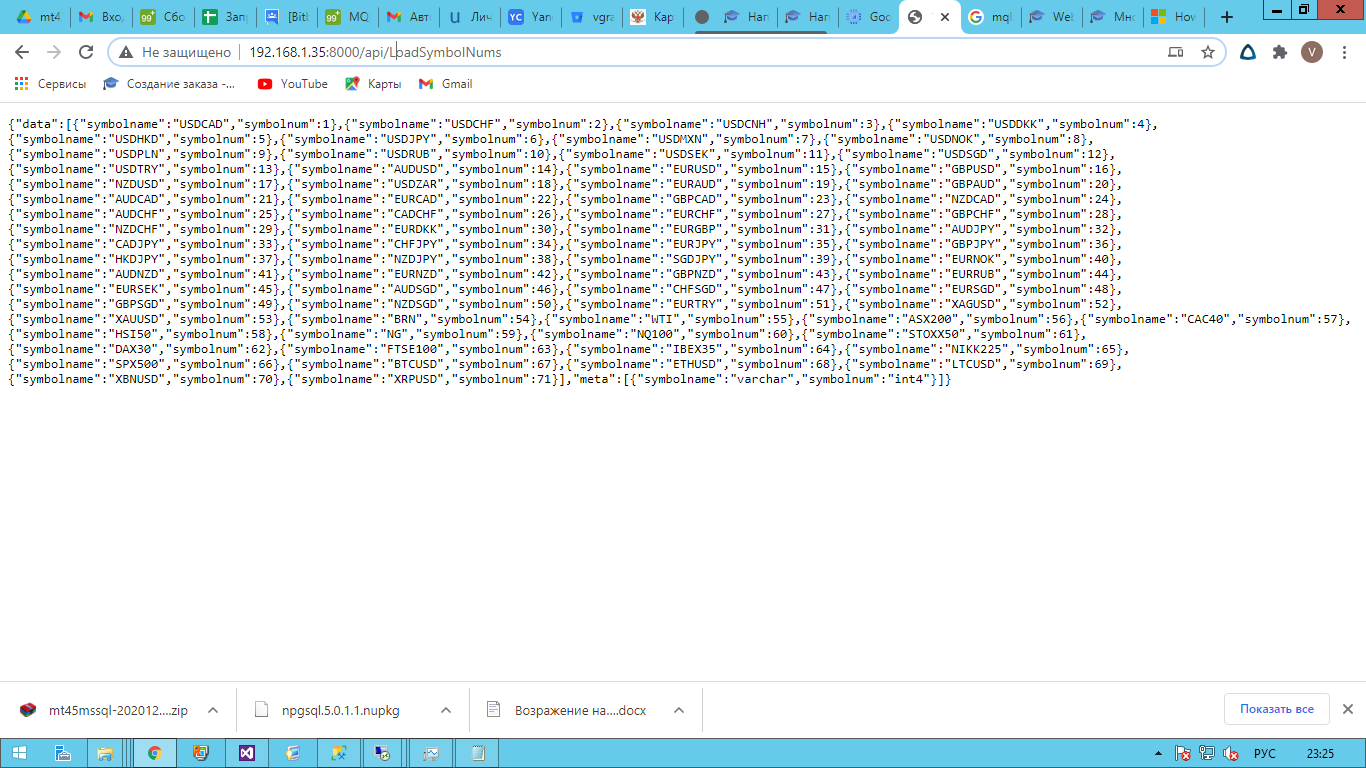
1. Цель: изменение библиотеки подключения экспертов к базе данных без изменения логики работы самих экспертов.
2. Краткое описание текущей реализации библиотеки на MQL5
   1. Библиотека является обверткой для работы DLL с хранимыми процедурами базы данных.
   2. Используется для работы 4-х независимых советников
3. Краткое описание сервиса
   1. Сервис развернут на локальной машине по адресу \_\_\_
   2. Наименование текущих функций подключения к БД и сервиса, а также передаваемые параметры идентичны.
   3. Сервис возвращает запрос в формате JSON. Во всех функциях кроме **LoadSymbolNums** ответ возвращается в виде одной строки передаваемых параметров.
   4. Пример запроса 
   5. Сервис генерации токена http://192.168.1.35:8000/api/get-token пока без авторизации раздает токены всем юзерам. Форма авторизации http://192.168.1.35:8000/Web/Auth
4. Список функций, подлежащих изменению

|  |  |
| --- | --- |
| Тип запроса | **Наименование функции** |
| Get | **int LoadSymbolNum (int AccountNumber, string symbolName, string &alert, int stype); //return symbolNum by symbolName** |
| Get | **int LoadSymbolNums(int AccountNumber, int &nums[], int maxcnt, char &names[], int buflen, int symlen, string &alert, int scriptType); //fill int &nums[], char &names[]** |
| set | **void UpdateInput (int AccountNumber, int SymbolNum, double Bid, double Ask);** |
| Get | **int GetSignal(int AccountNumber, int &ID, int &OrderTicket, int &Signal, int &Symbol\_Num, int &cmd, double &Volume, double &Price, int &Slippage, double &StopLoss, double &TakeProfit, int &comment, string &alert, int stype);** |
| Get | **int GetEmergencySignals(int AccountNumber, int &ID, int &OrderTicket, int &Signal, int &Symbol\_Num, int &cmd, double &Volume, double &Price, int &Slippage, double &StopLoss, double &TakeProfit, int &comment, string &alert, int stype);** |
| set | **int InsertTrades(int AccountNumber, int OrderTicket, int OrderTime, int OrderType, double OrderLots, int SymbolNum, double OrderOpenPrice, double OrderTakeProfit, double OrderStopLoss, double OrderProfit, int comment, string &alert, int stype);** |
| set | **int DeleteTrades(int AccountNumber, int OrderTicket, string &alert, int stype);** |
| set | **int InsertErrors(int AccountNumber, int IdSignal, int ErrorCode, string ErrorDescription, int SendCount, string &alert, int stype);** |
| Get | **long GetTradeLastUpdate(int AccountNumber, string &alert, int stype);** |
| set | **void UpdateAccounts(int AccountNumber, double AccountBalance, double AccountCredit, double AccountEquity, double AccountMargin, double AccountFreeMargin, string &alert, int stype);** |

1. Требования к WebRequest библиотеке
   1. Обмен запросами c веб сервисом
   2. Парсинг и передача параметров работы советникам
   3. В случае возникновения ошибок – передача почтовых сообщений на заданный e-mail
      1. Формат сообщения «Название процедуры» + Код ошибки
2. Пример текущей реализованной библиотеки

#ifndef MSSQLLIB

#define MSSQLLIB

//+------------------------------------------------------------------+

//| mssql.mqh |

//| Copyright 2019, MetaQuotes Software Corp. |

//| https://www.mql5.com |

//+------------------------------------------------------------------+

#property copyright "Copyright 2019, MetaQuotes Software Corp."

#property link "https://www.mql5.com"

#property strict

#define SYMBOLS\_MAX 100

#define SYMBOL\_MAXLEN 8

#define ERR\_INVALID\_TICKET 4108

//#define DBTRACE

#import "mt45mssql.dll"

int DBConnect(string HostName, int AccountNumber, string user, string password, string &alert, int scriptType);

int DBDisconnect (int AccountNumber, string &alert, int scriptType);

int LoadSymbolNum (int AccountNumber, string symbolName, string &alert, int stype); //return symbolNum by symbolName

int LoadSymbolNums(int AccountNumber, int &nums[], int maxcnt, char &names[], int buflen, int symlen, string &alert, int scriptType); //fill int &nums[], char &names[]

void UpdateInput (int AccountNumber, int SymbolNum, double Bid, double Ask);

//void TestUpdateInput(int AccountNumber, int SymbolNum, double Bid, double Ask, string &alert, int stype);

//void SetTestSignal(int AccountNumber,

int OrderTicket, int Signal, int Symbol\_Num, int cmd, double Volume, double Price, int Slippage, double StopLoss, double TakeProfit, int comment,

string &alert, int stype);

int GetSignal(int AccountNumber,

int &ID, int &OrderTicket, int &Signal, int &Symbol\_Num, int &cmd, double &Volume, double &Price, int &Slippage, double &StopLoss, double &TakeProfit, int &comment,

string &alert, int stype);

int GetEmergencySignals(int AccountNumber,

int &ID, int &OrderTicket, int &Signal, int &Symbol\_Num, int &cmd, double &Volume, double &Price, int &Slippage, double &StopLoss, double &TakeProfit, int &comment,

string &alert, int stype);

int InsertTrades(int AccountNumber, int OrderTicket, int OrderTime, int OrderType, double OrderLots,

int SymbolNum, double OrderOpenPrice, double OrderTakeProfit, double OrderStopLoss, double OrderProfit, int comment,

string &alert, int stype);

int DeleteTrades(int AccountNumber, int OrderTicket, string &alert, int stype);

int InsertErrors(int AccountNumber, int IdSignal, int ErrorCode, string ErrorDescription, int SendCount, string &alert, int stype);

long GetTradeLastUpdate(int AccountNumber, string &alert, int stype);

void UpdateAccounts(int AccountNumber, double AccountBalance, double AccountCredit, double AccountEquity, double AccountMargin, double AccountFreeMargin, string &alert, int stype);

#import

//--------------==========================-------------//

struct sSymbolNums

{

int num;

string name;

bool exist;

double prevBid, prevAsk;

};

sSymbolNums symbolNums[];

int getSymbolNum(string symbol)

{

for(int i=0; i<ArraySize(symbolNums); i++)

{

if(symbolNums[i].exist && symbol == symbolNums[i].name)

{

return symbolNums[i].num;

}

}

return -1;

}

uchar lnames[SYMBOLS\_MAX\*SYMBOL\_MAXLEN];

int lnums[SYMBOLS\_MAX];

//--------------==========================-------------//

struct dbAccount

{

int accountNumber;

string host, user, password;

int stype;

dbAccount(){accountNumber=0;}

};

dbAccount db\_;

//--------------==========================-------------//

struct sOrder

{

int id, ticket, signal, symbol\_num, cmd, slippage, comment;

double volume, price, SL, TP;

};

//sOrder order;

struct sPosition

{

int ticket;

datetime time;

int cmd;

double volume;

int symbol\_num;

double price, TP, SL, profit;

int comment;

};

//--------------==========================-------------//

enum eScriptType { ST\_input, ST\_trade, ST\_watch, ST\_test };

enum eCmd

{

cmd\_Enter = 100,

cmd\_Close = 200,

cmd\_CloseAllByComment = 225,

cmd\_CloseAllBySymbol = 250,

cmd\_CloseAll = 275,

cmd\_Modify = 300,

cmd\_400 = 400

};

//============================================================================//

//============================================================================//

void dbinit(eScriptType stype=ST\_test)

{

db\_.accountNumber = (int)AccountInfoInteger(ACCOUNT\_LOGIN);

db\_.stype = (int)stype;

ArrayResize(symbolNums, SYMBOLS\_MAX);

}

void dbinit2(string HostName, string user, string password)

{

db\_.accountNumber = (int)AccountInfoInteger(ACCOUNT\_LOGIN);

db\_.host = HostName;

db\_.user = user;

db\_.password = password;

ArrayResize(symbolNums, SYMBOLS\_MAX);

}

//============================================================================//

// Connect to database

// return values

// 0 -- connection success

// 1 -- already connected

//-1 -- error

int dbConnect(string HostName, string user, string password, string &alert)

{

if(db\_.accountNumber == 0) dbinit();

int res = DBConnect(HostName, db\_.accountNumber, user, password, alert, db\_.stype);

return res;

}

//============================================================================//

// Disconnect from database

// return values

// 0 -- OK

//-1 -- error

int dbDisconnect()

{

string alert="";

int res = DBDisconnect(db\_.accountNumber, alert, db\_.stype);

if(alert!="")

{

Print(alert);

string txt = TimeCurrent()+"\n"+alert;

SendMail("Error "+(string)AccountInfoInteger(ACCOUNT\_LOGIN)+" ",txt);

}

return res;

}

//============================================================================//

// заполняет массив структур symbolNums[]{int num; string name;}

// возвращает количество элементов в списке

int dbLoadSymbolNums()

{

string alert="";

int scnt = LoadSymbolNums(db\_.accountNumber, lnums, SYMBOLS\_MAX, lnames, SYMBOLS\_MAX\*SYMBOL\_MAXLEN, SYMBOL\_MAXLEN, alert, db\_.stype);

for(int i=0; i<scnt; i++)

{

symbolNums[i].name = CharArrayToString(lnames, i\*SYMBOL\_MAXLEN, SYMBOL\_MAXLEN);

symbolNums[i].num = lnums[i];

symbolNums[i].exist = false;

}

#ifdef DBTRACE

Print(scnt, " симоволов загружено");

for(int i=0; i<scnt; i++)

Print(symbolNums[i].num,":", symbolNums[i].name);

#endif

if(alert!="")

{

Print(alert);

string txt = TimeCurrent()+"\n"+alert;

SendMail("Error "+(string)AccountInfoInteger(ACCOUNT\_LOGIN)+" ",txt);

}

return scnt;

}

//============================================================================//

//return symbolNum by symbolName, -1 if symbolName not found

int dbLoadSymbolNum(string symbolName=NULL)

{

string alert="";

if(symbolName == NULL) symbolName = Symbol();

int res = LoadSymbolNum(db\_.accountNumber, symbolName, alert, db\_.stype);

if(alert!="")

{

Print(alert);

string txt = TimeCurrent()+"\n"+alert;

SendMail("Error "+(string)AccountInfoInteger(ACCOUNT\_LOGIN)+" ",txt);

}

return res;

}

//============================================================================//

void dbUpdateTick(int symbolNum, double Bid, double Ask)

{

string alert="";

UpdateInput(db\_.accountNumber, symbolNum, Bid, Ask);

if(alert!="")

{

Print(alert);

string txt = TimeCurrent()+"\n"+alert;

SendMail("Error "+(string)AccountInfoInteger(ACCOUNT\_LOGIN)+" ",txt);

}

//string alert="";

//TestUpdateInput(db\_.accountNumber, symbolNum, Bid, Ask, alert);

//Print("ex ", alert);

}

//============================================================================//

bool dbGetSignal(sOrder &order)

{

string alert="";

int res = GetSignal(db\_.accountNumber,

order.id, order.ticket, order.signal, order.symbol\_num,

order.cmd, order.volume, order.price, order.slippage,

order.SL, order.TP, order.comment, alert, db\_.stype);

if(alert!="")

{

Print(alert);

string txt = TimeCurrent()+"\n"+alert;

SendMail("Error "+(string)AccountInfoInteger(ACCOUNT\_LOGIN)+" ",txt);

}

return res > 0;

}

//============================================================================//

bool dbGetEmergencySignal(sOrder &order)

{

string alert="";

int res = GetEmergencySignals(db\_.accountNumber,

order.id, order.ticket, order.signal, order.symbol\_num,

order.cmd, order.volume, order.price, order.slippage,

order.SL, order.TP, order.comment, alert, db\_.stype);

if(alert!="")

{

Print(alert);

string txt = TimeCurrent()+"\n"+alert;

SendMail("Error "+(string)AccountInfoInteger(ACCOUNT\_LOGIN)+" ",txt);

}

return res > 0;

}

//============================================================================//

void dbInsertTrades(sPosition &pos)

{

string alert="";

int res = InsertTrades(db\_.accountNumber,

pos.ticket, (int)pos.time, pos.cmd, pos.volume, pos.symbol\_num,

pos.price, pos.TP, pos.SL, pos.profit,

pos.comment, alert, db\_.stype);

if(alert!="")

{

Print(alert);

string txt = TimeCurrent()+"\n"+alert;

SendMail("Error "+(string)AccountInfoInteger(ACCOUNT\_LOGIN)+" ",txt);

}

}

//============================================================================//

void dbDeleteTrades(sPosition &pos)

{

string alert="";

int res = DeleteTrades(db\_.accountNumber, pos.ticket, alert, db\_.stype);

if(alert!="")

{

Print(alert);

string txt = TimeCurrent()+"\n"+alert;

SendMail("Error "+(string)AccountInfoInteger(ACCOUNT\_LOGIN)+" ",txt);

}

}

//============================================================================//

void dbInsertErrors(int idSignal, uint retcode, string descript, int sendCnt)

{

string alert="";

int res = InsertErrors(db\_.accountNumber, idSignal, retcode, descript, sendCnt, alert, db\_.stype);

if(alert!="")

{

Print(alert);

// string txt = TimeCurrent()+"\n"+alert;

//SendMail("Error "+(string)AccountInfoInteger(ACCOUNT\_LOGIN)+" ",txt);

}

}

//============================================================================//

datetime dbGetTradeLastUpdate()

{

string alert="";

datetime tm = (datetime)GetTradeLastUpdate(db\_.accountNumber, alert, db\_.stype);

if(alert!="")

{

Print(alert);

string txt = TimeCurrent()+"\n"+alert;

SendMail("Error "+(string)AccountInfoInteger(ACCOUNT\_LOGIN)+" ",txt);

}

return tm;

}

//============================================================================//

void dbUpdateAccounts()

{

string alert="";

UpdateAccounts(db\_.accountNumber, AccountInfoDouble(ACCOUNT\_BALANCE),

AccountInfoDouble(ACCOUNT\_CREDIT),

AccountInfoDouble(ACCOUNT\_EQUITY),

AccountInfoDouble(ACCOUNT\_MARGIN),

AccountInfoDouble(ACCOUNT\_MARGIN\_FREE),

alert, db\_.stype);

if(alert!="")

{

Print(alert);

string txt = TimeCurrent()+"\n"+alert;

SendMail("Error "+(string)AccountInfoInteger(ACCOUNT\_LOGIN)+" ",txt);

}

}

//============================================================================//

//============================================================================//

//============================================================================//

#endif