```
//---------------------------------------------------------------------------

#include <Vcl.Dialogs.hpp>
#include <System.DateUtils.hpp>
#include <fstream>
#pragma hdrstop

#include "UnitDMClients.h"
#include "UnitFormMain.h"
#include "UnitUtils.h"
#include "UnitThreadWorking.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma classgroup "Vcl.Controls.TControl"
#pragma resource "*.dfm"
TdmClients *dmClients;
//---------------------------------------------------------------------------
#include <mutex>
std::mutex mutex;
//---------------------------------------------------------------------------
void __fastcall tsp::Client::Init(void)
{
    intLocalPort = -1;

    timeREQ = 0;
    timeCNC = 0;
    timeDSC = 0;
    timeRPL = 0;
    timeERR = 0;
    rpl = 0;
}
//---------------------------------------------------------------------------
void __fastcall tsp::ClientsPack::Init(void)
{
    for(int i = 0; i < vClients.size(); i++)
    {
        vClients[i]->Init();
    }

    dtStart = 0;
    dtEnd = 0;
}
//---------------------------------------------------------------------------
String __fastcall tsp::ClientsPack::ToMilliSeconds(TTime t, bool boolFormat)
{
    String strResult;

    if(!boolFormat)
    {   // hh:mm:ss.zzz
        strResult = FormatDateTime("hh:mm:ss.zzz", t);
    }
    else
    {
        // ms
        TTime t0;
        if(t != t0)
        {
            strResult = MilliSecondsBetween(t, vClients[0]->timeREQ);
        }
    }

    return strResult;
}
//---------------------------------------------------------------------------
void __fastcall tsp::ClientsPack::Save(String strFileName)
{
    // Resource Acquisition Is Initialization or RAII,
    // is a C++ programming technique[1][2] which binds
    // the life cycle of a resource that must be acquired before use
    // (allocated heap memory, thread of execution, open socket, open file,
    // locked mutex, disk space, database connection—anything
    // that exists in limited supply) to the lifetime of an object.
    std::wofstream file(strFileName.c_str());   // RAII
```

```
    if(file)
    {
        String strMode;
        if(boolConsecutive)
        {
            strMode = "Consecutive";
        }
        else
        {
            strMode = "Simultaneous";
        }

        file << "ТЕСТОВЕ ПОД ТОВАР НА TIME PROTOCOL (RFC-868) СЪРВЪР" << std::endl << std::endl;

        file << "Източник на товар:;;;;" << formMain->Caption << std::endl << std::endl;

        file << "Дата:;;;;" << FormatDateTime("yyyy.mm.dd", dtStart) << std::endl;
        file << "IP адрес сървър:;;;;" << strIPAddress << std::endl;
        file << "IP адрес клиент:;;;;" << vClients[0]->csOut->Socket->LocalAddress << std::endl;
        file << "Обем на пакета:;;;;" << intVolume << std::endl;
        file << "Режим на формиране на заявките:;;;;" << strMode << std::endl;
        file << "Задръжка до разпадане:;;;;" << intDelayToFree << std::endl << std::endl;

        file << "Начало:;;;;" << FormatDateTime("hh:mm:ss.zzz", dtStart) << std::endl;
        file << "Край:;;;;" << FormatDateTime("hh:mm:ss.zzz", dtEnd) << std::endl;
        file << "Продължителност:;;;;" << FormatDateTime("s.zzz", dtEnd - dtStart) << std::endl << s

        file << "Client;T REQ;T CNC;T RPL;T DSC;T ERR;Reply;Port" << std::endl;

        for(int i = 0; i < vClients.size(); i++)
        {
            file << i << ";"
                 << ToMilliSeconds(vClients[i]->timeREQ, true) << ";"
                 << ToMilliSeconds(vClients[i]->timeCNC, true) << ";"
                 << ToMilliSeconds(vClients[i]->timeRPL, true) << ";"
                 << ToMilliSeconds(vClients[i]->timeDSC, true) << ";"
                 << ToMilliSeconds(vClients[i]->timeERR, true) << ";"
                 << std::hex << std::uppercase << vClients[i]->rpl << ";"
                 << std::dec << vClients[i]->intLocalPort
                 << std::endl;
//               << std::endl;
        }

        MessageDlg(strFileName + " saved", mtInformation, TMsgDlgButtons() << mbOK, 0);
    }
    else
    {
        throw Exception("Cannot open " + strFileName);
    }
}
//---------------------------------------------------------------------------
void __fastcall tsp::Log::Add(String str)
{
    String ws = FormatDateTime("hh:mm:ss.zzz",Time()) + " " + str;

    // формиране на критична секция до изхода на функцията
    // секцията сериализира достъпа на нишките до GUI
    std::lock_guard<std::mutex> guard(mutex);   // RAII
    //
    Lines.push_back(ws);
}
//---------------------------------------------------------------------------
void __fastcall tsp::Log::Add(TCustomWinSocket* sock, String str)
{
    String ws = FormatDateTime("hh:mm:ss.zzz",Time()) +
                " [" + sock->RemoteHost + "::" + sock->RemoteAddress + "] " +
                str;

    // формиране на критична секция до изхода на функцията
    // секцията сериализира достъпа на нишките до GUI
    std::lock_guard<std::mutex> guard(mutex);   // RAII
    //
    Lines.push_back(ws);
}
```

```
//---------------------------------------------------------------------------
void __fastcall tsp::Log::Show(TMemo* dst)
{
    for(int i = 0; i < Lines.size(); i++)
    {
        dst->Lines->Add(Lines[i]);
    }
}
//---------------------------------------------------------------------------
__fastcall TdmClients::TdmClients(TComponent* Owner)
    : TDataModule(Owner)
{
    __read_ini_file();  // четене на параметрите от конфигурационния файл

    for(int i = 0; i < clientsPack.intVolume; i++)
    {
        tsp::Client* tspClient = new tsp::Client;

        tspClient->csOut = new TClientSocket(Owner);
        tspClient->csOut->Port = 37;
        tspClient->csOut->Address = clientsPack.strIPAddress;

        tspClient->csOut->OnConnect = csOut->OnConnect;
        tspClient->csOut->OnDisconnect = csOut->OnDisconnect;
        tspClient->csOut->OnError = csOut->OnError;
        tspClient->csOut->OnRead = csOut->OnRead;

        tspClient->State = tsp::Client::Closed;

        clientsPack.vClients.push_back(tspClient);
    }
    clientsPack.intPending = 0;
    clientsPack.boolSyncing = false;
}
//---------------------------------------------------------------------------
__fastcall TdmClients::~TdmClients(void)
{
    __write_ini_file();
}
//---------------------------------------------------------------------------
void __fastcall TdmClients::__read_ini_file(void)
{
    strIniFileName = Application->ExeName;
    strIniFileName = strIniFileName.SubString(1, strIniFileName.Length() - 3) + "ini";

    iniFile = new TIniFile(strIniFileName);
    if(!FileExists(strIniFileName))
    {
        MessageDlg("Missing configuration file!", mtError, TMsgDlgButtons() << mbOK, 0);
    }

    formMain->Top   = iniFile->ReadInteger(L"FormPos", L"Top", 0);
    formMain->Left  = iniFile->ReadInteger(L"FormPos", L"Left", 0);

    formMain->memoLog->Visible = !iniFile->ReadInteger(L"Log", L"Visible", 0);

    clientsPack.strIPAddress    = iniFile->ReadString(L"TimeServer", L"IPAddress", L"127.0.0.1");

    clientsPack.intVolume       = iniFile->ReadInteger(L"Package", L"Volume", 1);
    clientsPack.intDelayToFree  = iniFile->ReadInteger(L"Package", L"DelayToFree", 0);
    clientsPack.boolConsecutive = iniFile->ReadInteger(L"Package", L"Consecutive", 1);
}
//---------------------------------------------------------------------------
void __fastcall TdmClients::__write_ini_file(void)
{
    try
    {
        iniFile->WriteInteger("FormPos", "Top", formMain->Top);
        iniFile->WriteInteger("FormPos", "Left", formMain->Left);

        iniFile->WriteInteger(L"Log", L"Visible", formMain->memoLog->Visible);
    }
    catch(Exception& e)
    {
```

```
        MessageDlg(e.Message, mtError, TMsgDlgButtons() << mbOK, 0);
    }

    delete iniFile;
}
//---------------------------------------------------------------------------
int __fastcall TdmClients::GetChannelId(TObject *Sender)
{
    int intId = -1;

    for(int i = 0; i < clientsPack.vClients.size(); i++)
    {
        if(clientsPack.vClients[i]->csOut == Sender)
        {
            intId = i;
            break;
        }
    }

    return intId;
}
//---------------------------------------------------------------------------
void __fastcall TdmClients::SendBatchOfReq(void)
{
    if(clientsPack.boolSyncing)
    {
        MessageDlg("Cannot start nested sync", mtError, TMsgDlgButtons() << mbOK, 0);
        return;
    }

    if(formMain->memoLog->Visible)
    {
        formMain->labelShowHideLogClick(formMain);
    }
    formMain->memoLog->Clear();

    formMain->buttonSync->Enabled = false;
    Screen->Cursor = crHourGlass;

    log.Clear();

    clientsPack.intPending = clientsPack.intVolume;
    clientsPack.boolSyncing = true;
    clientsPack.Init();

    if(clientsPack.boolConsecutive)
    {
        if(clientsPack.vClients.size() > 0)
        {
            clientsPack.vClients[0]->csOut->Open();
            clientsPack.vClients[0]->State = tsp::Client::Transient;
            clientsPack.vClients[0]->intLocalPort = clientsPack.vClients[0]->csOut->Socket->LocalPo
            clientsPack.vClients[0]->timeREQ = Time();
        }
    }
    else
    {
        for(int i = 0; i < clientsPack.vClients.size(); i++)
        {
            clientsPack.vClients[i]->csOut->Open();
            clientsPack.vClients[i]->State = tsp::Client::Transient;
            clientsPack.vClients[i]->intLocalPort = clientsPack.vClients[i]->csOut->Socket->LocalPo
            clientsPack.vClients[i]->timeREQ = Time();
        }
    }

    clientsPack.dtStart = Date() + clientsPack.vClients[0]->timeREQ;
}
//---------------------------------------------------------------------------
void __fastcall TdmClients::csOutConnect(TObject *Sender, TCustomWinSocket *Socket)
{
    int intReqId = GetChannelId(Sender);
    log.Add(Socket, "CNC[" + IntToStr(intReqId) + "]");
```

```
    if(intReqId >= 0)
    {
        clientsPack.vClients[intReqId]->State = tsp::Client::Open;
        clientsPack.vClients[intReqId]->timeCNC = Time();

        if(clientsPack.boolConsecutive)
        {
            if(++intReqId < clientsPack.vClients.size())
            {
                clientsPack.vClients[intReqId]->csOut->Open();
                clientsPack.vClients[intReqId]->State = tsp::Client::Transient;
                clientsPack.vClients[intReqId]->intLocalPort = clientsPack.vClients[intReqId]->csOu
                clientsPack.vClients[intReqId]->timeREQ = Time();
            }
        }
    }
}
//---------------------------------------------------------------------------
void __fastcall TdmClients::csOutDisconnect(TObject *Sender, TCustomWinSocket *Socket)
{
    int intReqId = GetChannelId(Sender);
    log.Add(Socket, "DSC[" + IntToStr(intReqId) + "]");

    if(intReqId >= 0)
    {
        clientsPack.intPending--;
        clientsPack.vClients[intReqId]->State = tsp::Client::Closed;
        clientsPack.vClients[intReqId]->timeDSC = Time();
    }

    if(clientsPack.intPending == 0)
    {
        clientsPack.boolSyncing = false;
        clientsPack.dtEnd = Date() + clientsPack.vClients[intReqId]->timeDSC;

        log.Show(formMain->memoLog);
        if(!formMain->memoLog->Visible)
        {
            formMain->labelShowHideLogClick(formMain);
        }
        formMain->buttonSync->Enabled = true;
        Screen->Cursor = crDefault;
    }
}
//---------------------------------------------------------------------------
void __fastcall TdmClients::csOutError(TObject *Sender, TCustomWinSocket *Socket,
          TErrorEvent ErrorEvent, int &ErrorCode)
{
    int intReqId = GetChannelId(Sender);
    log.Add(Socket, "ERR[" + IntToStr(intReqId) + "][" + IntToStr(ErrorCode) + "]");

    if(intReqId >= 0)
    {
        clientsPack.intPending--;
        clientsPack.vClients[intReqId]->State = tsp::Client::Closed;
        clientsPack.vClients[intReqId]->timeERR = Time();
    }

    if(clientsPack.intPending == 0)
    {
        clientsPack.boolSyncing = false;
        clientsPack.dtEnd = Date() + clientsPack.vClients[intReqId]->timeERR;

        log.Show(formMain->memoLog);
        if(!formMain->memoLog->Visible)
        {
            formMain->labelShowHideLogClick(formMain);
        }
        formMain->buttonSync->Enabled = true;
        Screen->Cursor = crDefault;
    }

    ErrorCode = 0;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TdmClients::csOutRead(TObject *Sender, TCustomWinSocket *Socket)
{
    unsigned long ulTime;

    Socket->ReceiveBuf(&ulTime, 4);
    ulTime = ntohl(ulTime);

    int intReqId = GetChannelId(Sender);
    if(intReqId >= 0)
    {
        clientsPack.vClients[intReqId]->timeRPL = Time();
        clientsPack.vClients[intReqId]->rpl = ulTime;
    }
    log.Add(Socket, "RPL[" + IntToStr(intReqId) + "]::[" + IntToHex((int)ulTime, 8) + "]");

    // СИМУЛАЦИЯ НА ОБРАБОТКА
    // многозадачно обслужване за избягване на сериализацията
    // на паралелните клонове
    //
    std::thread threadWorking(DoWork, Socket, intReqId, clientsPack.intDelayToFree);
    threadWorking.detach();     // развързване на дъщерната нишка от основната
    ////////////////////////////////////////////////////////////////////////
}
//---------------------------------------------------------------------------
```