1.1 Ziel des Projektes

- Grundlegender Umgang mit dem Socket-API.
- Programmierung einer Client-Anwendung, die vom Zeitserver (141.55.192.51) die Zeit abfragt und diese in lesbarer Form darstellt.
- Der Zeitserver arbeitet entsprechend RFC 868.
- Verschaffen Sie sich zunächst anhand des RFC einen Überblick, wie dieser Server arbeitet.
- Verschaffen Sie sich mittels der Hilfefunktion unter Borland-CPP-Builder einen Überblick zu den Zeit/Datumsfunktionen asctime(), localtime()

1.2 Grundlagen

Ein nach RFC 868 arbeitender Zeitserver, liefert die vergangenen Sekunden seit dem 1.1.1900. Er stellt diese in 4 Bytes (LongWord) in der so genannten Network-Order (big-endian) zur Verfügung.
Damit kann der Server bis $\mathbf{2^{32} = 4.294.967.296}$ zählen.

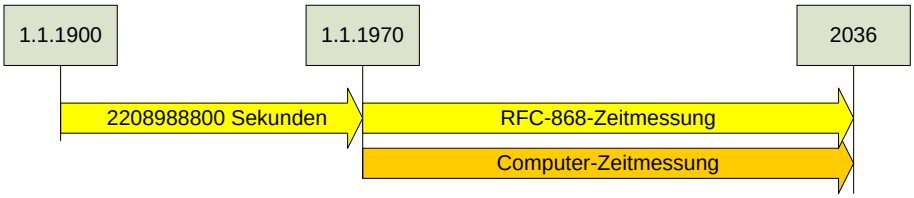
✎ Wie viel Jahre kann man diesen Zeitserver nutzen?

$$Anzahl_{Jahre} = \frac{4294967296}{s*m*h*d} = \frac{4294967296}{60*60*24*365} = \frac{4294967296}{3024000} \approx \underline{\underline{136{,}19}}$$

Dieser Zeitserver ist damit rund von 1900 bis zum Jahr 2036 nutzbar.

Viele Funktionen zur Datumsberechnung in Computern basieren aber auf einer Zählung der Sekunden ab 1.1.1970. Deshalb muss man, wenn man Computer-Datumsfunktionen nutzen will, diesen Zeitstempel auf den 1.1.1970 normalisieren. Von dem Zeitstempel des Time-Servers muss man deshalb 2208988800 Sekunden abziehen (70 Jahre a 365 Tage + 17 Tage dazu wegen der Schaltjahre):

s*m*h*365*70 + s*m*h*17= 60*60*24*365*70 + 60*60*24*17 = 2208988800



Auszug aus RFC 868 Time Protocol

```
This protocol may be used either above the Transmission Control Protocol
(TCP) or above the User Datagram Protocol (UDP).
When used via TCP the time service works as follows:
   S: Listen on port 37 (45 octal).

   U: Connect to port 37.

   S: Send the time as a 32 bit binary number.

   U: Receive the time.

   U: Close the connection.

   S: Close the connection.

   The server listens for a connection on port 37.  When the connection
   is established, the server returns a 32-bit time value and closes the
   connection.  If the server is unable to determine the time at its
   site, it should either refuse the connection or close it without
   sending anything.
```

localtime

Header-Datei `<time.h >`
Kategorie Uhrzeit- und Datumsroutinen
Prototyp `struct tm *localtime(const time_t *timer);`
Beschreibung Konvertiert Datum und Zeit in eine Struktur. *localtime* akzeptiert die Adresse eines von *time* zurückgegebenen Werts und gibt einen Zeiger auf eine Struktur des Typs *tm* zurück, die Zeitwerte enthält. Die Funktion berücksichtigt die Zeitzone und gegebenenfalls die Sommerzeit.

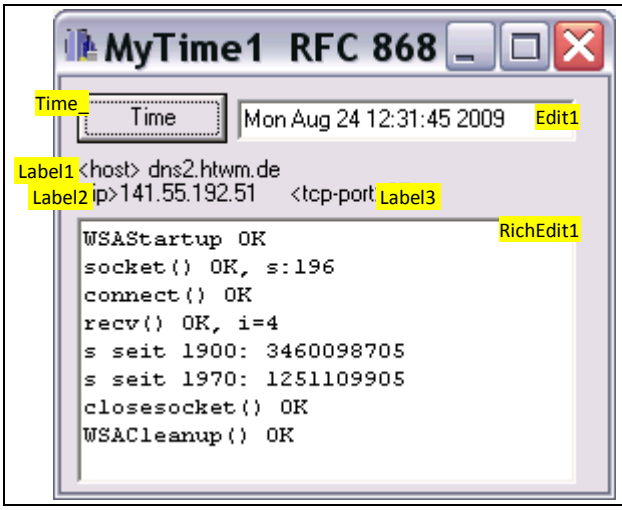
Die Struktur *tm* ist in der Header-Datei time.h folgendermaßen deklariert:

```
struct tm { int tm_sec; int tm_min; int tm_hour; int tm_mday; int tm_mon; int tm_year;
int tm_wday; int tm_yday; int tm_isdst; };
```

asctime

Header-Datei `<time.h >`
Kategorie Uhrzeit- und Datumsroutinen
Prototyp `char *asctime(const struct tm *tptr);`
The function stores in the static-duration time string a 26-character English-language representation of the time encoded in *tptr. It returns the address of the static-duration time string. The text representation takes the form:
`Sun Dec  2 06:55:15 1979\n\0`

1.3 Realisierung des Projektes MyTime1_RFC868_WINSOCK_STREAM



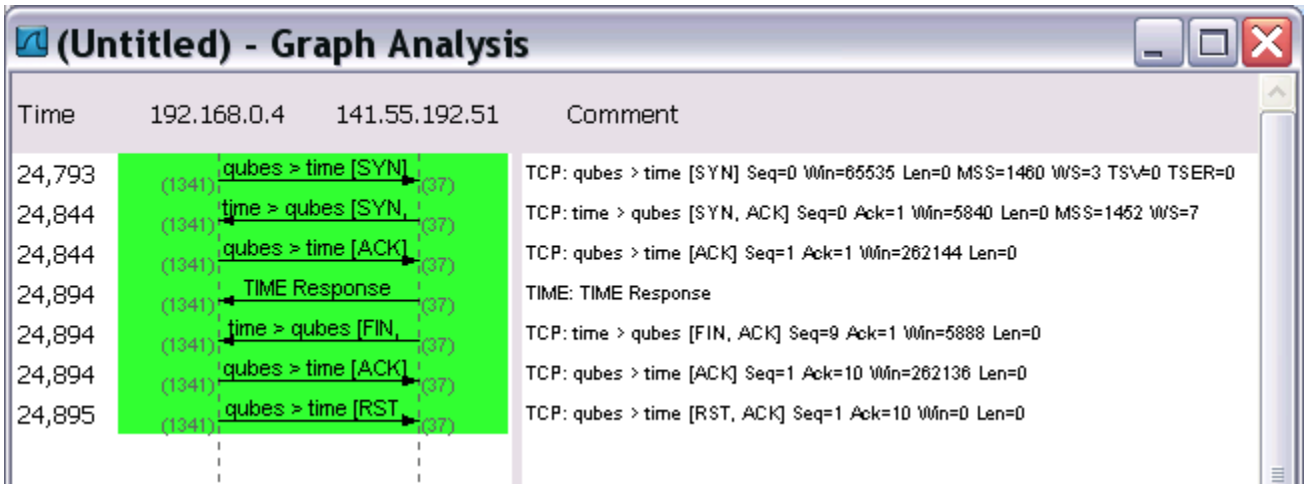
- Erzeugen Sie ein neues Projekt " **MyTime1_RFC868_WINSOCK_STREAM**" im gleichnamigen Order.
- Editieren Sie die Oberfläche entsprechend der nebenstehenden Abbildung.
- Erzeugen Sie Schritt für Schritt den Programmcode, entsprechend dem Beispiel-Code.
- Nutzen Sie die IP-Adresse: 141.155.192.151[1], Port 37

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <winsock.h>//erforderlich
#include <time.h>   //erforderlich
#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
            : TForm(Owner)
{
}
//---------------------------------------------------------------------------
```

[1] Der Hostname lautet dns2.htwm.de. Dort läuft neben dem Dienst "DNS" (Port 53), auch noch der Dienst "Time Protocol" (Port 37)

```
void __fastcall TForm1::Time_Click(TObject *Sender)
{
       RichEdit1->Clear();
//===(1) Am API anmelden; nur bei WINSOCK
//Funktion: WSAStartup(); Variable: WSADATA wsaData;
       WSADATA wsaData;
       if (WSAStartup(0x101,&wsaData))
        {RichEdit1->Lines->Add("WSAStartup ERR");goto ende;}
       else
        RichEdit1->Lines->Add("WSAStartup OK");

//===(2) Socket errichten
//Funktion: socket(); Merke: s =====
       int s;//Socket-Descriptor
       s=socket(AF_INET,SOCK_STREAM,0);
       if (s==0)
        {RichEdit1->Lines->Add("socket() ERR"); goto ende;}
       else
        RichEdit1->Lines->Add("socket() OK, s:"+IntToStr(s));

//===(3) Hostadresse (IP, Port) eintragen, wohin die Verbindung gehen soll
//Variable: SOCKADDR_IN sAddr ===
       SOCKADDR_IN sAddr;
       sAddr.sin_family=AF_INET;
       sAddr.sin_port=htons(37);
       sAddr.sin_addr.S_un.S_addr=inet_addr("141.55.192.51");

//===(4) Verbindung herstellen
//Funktion: connect()  ===
       if (connect(s,(struct sockaddr *) &sAddr,sizeof(sAddr))!=0)
        {RichEdit1->Lines->Add("connect() ERR"); goto ende;}
       else
        RichEdit1->Lines->Add("connect() OK");

//===(5) Daten aus Socket lesen
//Funktion: recv();  ===
       int i; time_t t;
        i=recv(s, (char *) &t,4,0);
       if (i!=4)
        {RichEdit1->Lines->Add("recv() ERR"); goto ende;}
       else
        {RichEdit1->Lines->Add("recv() OK, i="+IntToStr(i));
         RichEdit1->Lines->Add("s seit 1900: "+(AnsiString)ntohl(t) );
         t=ntohl(t)-2208988800;
         RichEdit1->Lines->Add("s seit 1970: "+(AnsiString)t);
         Edit1->Text=((AnsiString)asctime(localtime(&t))).SubString(0,24);
        }
//===(6) Verbindung beenden
//Funktion: closesocket();  ===
        if (closesocket(s)!=0)
         {RichEdit1->Lines->Add("closesocket() ERR"); goto ende;}
        else
         RichEdit1->Lines->Add("closesocket() OK");
//===(7) Am API abmelden; nur bei WINSOCK
//Funktion: WSACleanup();  ===
       if (WSACleanup()!=0)
         {RichEdit1->Lines->Add("WSACleanup() ERR"); goto ende;}
        else
         RichEdit1->Lines->Add("WSACleanup() OK");
ende:
}
//---------------------------------------------------------------------------
```

1.4 Wireshark-Trace



0000	00 18 39 15 f4 fb 00 1f 3f ad d7 1e 08 00 45 00	MAC
0010	00 30 46 e6 40 00 38 06 ed ca 8d 37 c0 33 c0 a8	IP
0020	00 04 00 25 05 3d 42 c5 75 43 0d c5 75 e4 50 18	TCP
0030	00 2e a7 9c 00 00 ce 3c ea 91 00 00 00 00	*Timeprotokoll*