```
//Latest update: 30.05.2017
//Latest improvement: Timer added
const float version = 1.0;
/*
   Arduino_TCP

   A simple TCP Server that allows to talk to the Arduino via Telnet.
   It can process the given commands and execute them.

   This script is written for the Arduino Leonardo ETH.

   //TELNET-Version

*/

//========== Needed Packages ==========
#include<EEPROM.h>
#include <SPI.h>
#include <Ethernet2.h> //Maybe this package needs to be changed
#include <MsTimer2.h> //Needed to start a timer

//========== Define Constants ==========
#define MAX_CMD_LENGTH 25
// Note: All pins are at startup by default in INPUT-Mode (besides
PIN 13!!)
#define testled 13 //Select testled pin

//==========Static/DHCP==============
byte mac[] = {0x90, 0xA2, 0xDA, 0x10, 0xC3, 0x0F}; // Enter a MAC
address

struct IPObject {
   boolean dhcp_state; //static IP (false) or DHCP (true)
   int ip_addr[4]; //IP address
};

IPObject ip_eeprom;
int eeAddress = 0; // Defined address for EEPROM

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

//========== Ethernet server port ==========
```

```
// Initialize the Ethernet server library with a port you want to use
EthernetServer server(23); //TELNET defaults to port 23


//========== Needed variables or constants ==========
boolean connected = false; // whether or not the client was connected
previously
String cmdstr; // command string that will be evaluated when received
via ethernet
const byte readonstate[]  = { A0, A2, A4 }; // Define analog pins in
array for on measurement of valve status
const byte readoffstate[] = { A1, A3, A5 }; // Define analog pins in
array for off measurement of valve status
//===== Timer variables and constants =====
boolean allowchange = false; // whether or not the status of a valve
may be changed
const uint16_t timeperiod = 2000; // timer to wait until valve status
may be changed in ms

//===== Pins needed for slit control =====
#define lasershutterpin 0 //Select pin for lasershutter

//===== Pins needed for slit control =====
#define slit_ZLDP210P_on 11 // slit pressure ZLDP210P "Endschalter 1"
#define slit_ZLDP210P_off 12 // slit pressure ZLDP210P "Endschalter 2"

void timer(){
    allowchange = true;
}


void setup() {
    //===== Serial =====
   Serial.begin(9600); // Open serial communications and wait for
port to open

    //===== Ethernet =====
   EEPROM.get(eeAddress, ip_eeprom); // Get last written information
about DHCP or static IP and save it to ip_eeprom
   if (ip_eeprom.dhcp_state)
       Ethernet.begin(mac); // start the Ethernet connection and the
server using DHCP
```

```
    else {
       byte ip[4];
       for (int i = 0; i <= 3; i++) ip[i] = ip_eeprom.ip_addr[i];
      Ethernet.begin(mac, ip); // start the Ethernet connection and
the server using static IP written to EEPROM
    }

   server.begin(); // start server

   //===== Define Input and Output pins =====
   pinMode(testled, OUTPUT); // set led pin to output
   pinMode(lasershutterpin, OUTPUT); // set led pin to output
    for (int i = 1; i<=8; i++) {
       pinMode(i,OUTPUT); // set pin i to output
       digitalWrite(i,HIGH); // set pin i to HIGH (inverse logic for
relais)
    }
    for (int i = 0; i <= 2; i++) {
      digitalWrite(readonstate[i], INPUT_PULLUP); // Set pins for
on state to input_pullup
      digitalWrite(readoffstate[i], INPUT_PULLUP); // Set pins for
off state to input_pullup
    }
   //==== Do the same for the pneumatic slit driver ====
   //Endschalter on pin 11 and 12
  digitalWrite(slit_ZLDP210P_on, INPUT_PULLUP); // Set pins for on
state to input_pullup
  digitalWrite(slit_ZLDP210P_off, INPUT_PULLUP); // Set pins for
off state to input_pullup

   //==== Timer =====
    MsTimer2::set(timeperiod, timer); //
    MsTimer2::start();

   //===== Serial communication on startup =====
   Serial.print("server is at ");
  Serial.println(Ethernet.localIP());

}


void loop() {
```

```
   //===== Ethernet: Create a client connection =====
   EthernetClient client = server.available();
   EEPROM.get(eeAddress, ip_eeprom); // Get last written information
about DHCP or static IP and save it to ip_eeprom

   //==== Evaluate Serial input ====
   if (Serial.available() > 0) {
      String serialstring = Serial.readString(); // read the
incoming byte:
     serialstring.trim(); //removes leading and trailing whitespace
      Serial.println(serialstring); // write the read String

      if (serialstring.equals("help")) {
         Serial.println("--- Serial connection Help ---");
          Serial.println("ip?                : get current ip
address and configuration");
          Serial.println("set ip 10.32.200.44 : set static ip
address");
          Serial.println("ip: DHCP           : set ip to DHCP");
       }

      else if (serialstring.equals("ip?")) {
         Serial.print("server is currently at ");
Serial.println(Ethernet.localIP());
         Serial.print("ip_eeprom.dhcp_state: ");
Serial.print(ip_eeprom.dhcp_state);
          if (ip_eeprom.dhcp_state)
             Serial.println("(DHCP)");
           else {
             Serial.println("(static)"); Serial.print("EEPROM ip
");
              for (int i = 0; i <= 2;i++){
                    Serial.print(ip_eeprom.ip_addr[i]);
Serial.print(".");
                    if (i==2) Serial.println(ip_eeprom.
ip_addr[3]);
                  }
           }
       }

      else if (serialstring.startsWith("set ip ")) {
```

```
          char achar[23]; // Needed to use sscanf with input
serialstring
          int intarray[4]; // Intermediate integer array needed to
use sscanf

         serialstring.toCharArray(achar, sizeof(achar)); // Needed
to use sscanf with String a
             //Error!//sscanf(achar, "set ip %3d.%3d.%3d.%3d",
&ip_eeprom.ip_addr[0],&ip_eeprom.ip_addr[1],&ip_eeprom.ip_addr[3],
&ip_eeprom.ip_addr[4]); // Write new ip to ip_eeprom //Leads to error!
          sscanf(achar, "set ip %3d.%3d.%3d.%3d", &intarray[0],
&intarray[1], &intarray[2], &intarray[3]); //Assign new ip to
intermediate array          Serial.print("read ip: ");
           for (int i = 0; i <= 3; i++) {
              ip_eeprom.ip_addr[i] = intarray[i]; // Assign read ip
to EEPROM structure
            }
           for (int i = 0; i <= 2; i++) {
             Serial.print(ip_eeprom.ip_addr[i]); Serial.print(".");
               if (i == 2) {
                 Serial.print(ip_eeprom.ip_addr[3]); Serial.
println();
                  Serial.println("IP was written to EEPROM (restart
necessary)");
                }
            }
          ip_eeprom.dhcp_state = false; // Set IP dhcp_state to 0
for static
          EEPROM.put(eeAddress, ip_eeprom);
           }

      else if (serialstring.equals("ip: DHCP")) {
          Serial.println("Unset static ip, set DHCP (restart
necessary)");
          ip_eeprom.dhcp_state = true; // Set IP dhcp_state to 1
for DHCP
          EEPROM.put(eeAddress, ip_eeprom);
        }

      else Serial.println("Invalid command, type help");
    }
```

```
   //==== Ethernet ====
   if (client) {
      if (!connected) {
         client.flush(); //Clear out the input buffer
         cmdstr = ""; //clear the commandString variable
          connected = true;
       }

      if (client.available() > 0) {
          char zeichen = client.read();
         //convert all capitalized letters into small ones
          if ((65 <= zeichen) && (zeichen <= 90))
              zeichen += 32;
         //Filter all unwanted characters and first character must
not be space
          //only allowed: \n * : . ' ' ? a..z A..Z 0..9
           if (
             (zeichen == '\n') || (zeichen == '*') || (zeichen ==
':') || (zeichen == '.') || ((zeichen == ' ') && cmdstr.length()) ||
             (zeichen == '?') || ((48 <= zeichen) && (zeichen <=
57)) || ((97 <= zeichen) && (zeichen <= 122))
            )
              readTelnetCommand(zeichen, client);
       }
    }

}


void readTelnetCommand(char c, EthernetClient &client) {
   if (cmdstr.length() == MAX_CMD_LENGTH) {
       cmdstr = "";
    }

   if (c == '\n')
      parseCommand(client);
   else
       cmdstr += c;
   //====Debug====
   //Serial.print("cmdstr_read = ");
   //Serial.println(cmdstr);
```

```
}

voidparseCommand(EthernetClient &client) {

   //====Debug====
   //Serial.print("cmdstr = ");
  //Serial.println(cmdstr);

   //===== QUIT =====
   if (cmdstr.equals("quit")) {
       client.stop();
       connected = false;
    }

   //===== HELP =====
   else if (cmdstr.equals("help")) {
      client.print("--- Abacus Valve Version ");client.
print(version,1);client.println(" ---");
       client.println("on|off                  : switch test led
on/off");
       client.println("quit                    : close the
connection");
       client.println("ip?                     : get ip address");
      client.println("ch {1|2..|8} {off|on|?}  : set channel {1|2..
|8}{off|on|?}");
       client.println("meas:ch {1|2|3}         : read channel
{1|2|3} state");
       client.println("ls {off|on|?}           : open or close
lasershutter");
       client.println("slit ?                  : read current slit
status ");
       client.println();
      client.println("By S.Friederich");
    }

   //===== IP =====
   else if (cmdstr.equals("ip?")) {
      client.println(Ethernet.localIP());
      //Debug: Serial.println(Ethernet.localIP());
    }
```

```
   //===== TestLED =====
   else if (cmdstr.equals("on")) {
      digitalWrite(testled, HIGH);
    }
   else if (cmdstr.equals("off")) {
      digitalWrite(testled, LOW);
    }

   //===== Read Test-LED state on/off =====
   else if(cmdstr.equals("tled?")) {
      client.print("tled:");
     client.println(digitalRead(testled));
    }

   //===== Set valve channel on|off =====
   else if (cmdstr.startsWith("ch ")) {
      int channelnumber = cmdstr.substring(3,4).toInt(); // e.g.
"ch 1 on"
       if (cmdstr.endsWith("on")) {
          if (allowchange == true && digitalRead(channelnumber) ==
HIGH) {
              digitalWrite(channelnumber, LOW); // Inverse logic
(on=LOW)
               allowchange = false;
               MsTimer2::start();
            }
        }
       else if (cmdstr.endsWith("off")) {
          if (allowchange == true && digitalRead(channelnumber) ==
LOW) {
              digitalWrite(channelnumber, HIGH); // Inverse logic
(on=LOW)
               allowchange = false;
               MsTimer2::start();
            }
        }
       //===== Read set channel state =====
       else if (cmdstr.endsWith("?")) {
          boolean channelstate = !digitalRead(channelnumber); //
Inverse logic (1=on=LOW)
          client.print("current set state: ch ");client.
print(channelnumber);
```

```
        client.print(" ");client.println(channelstate);
      }
      else {
        client.println("Command invalid");
      }

   }

  //===== Read actual valve channel state =====
  else if (cmdstr.startsWith("meas:ch ")) {
     int channelnumber = cmdstr.substring(8,9).toInt(); // e.g.
"meas:ch 1"
      if (channelnumber > 3) // Currently: only 3 channel states
are wired and assigned
         client.println("Invalid channel");
      else {
          int channelstate_on =
digitalRead(readonstate[channelnumber-1]); //
          int channelstate_off =
digitalRead(readoffstate[channelnumber-1]); //
             client.print("actual state: ch ");client.
print(channelnumber);client.print(" ");
             if (channelstate_on == LOW && channelstate_off)
                 client.println("1");
             else if (channelstate_on && channelstate_off == LOW)
                 client.println("0");
              else
                 client.println("undefined");
      }
   }

  //===== Laser Shutter =====
  else if (cmdstr.equals("ls on"))
     digitalWrite(lasershutterpin, HIGH);
  else if (cmdstr.equals("ls off"))
     digitalWrite(lasershutterpin, LOW);

  //===== Read Laser Shutter state on/off =====
  else if(cmdstr.equals("ls ?")) {
      client.print("ls: ");
    client.println(digitalRead(lasershutterpin));
   }
```

```
  //===== Read Slit ZLDP210P state on/off =====
  else if(cmdstr.equals("slit ?")) {
     int channelstate_on = digitalRead(slit_ZLDP210P_on); //
     int channelstate_off = digitalRead(slit_ZLDP210P_off); //
      client.print("actual slit state: ");
             if (channelstate_on == LOW && channelstate_off)
                 client.println("1");
             else if (channelstate_on && channelstate_off == LOW)
                 client.println("0");
              else
                 client.println("undefined");
   }


  //===== Invalid Command, HELP =====
   else {
     client.println("Invalid command, type help");
   }

   cmdstr = "";
}
```