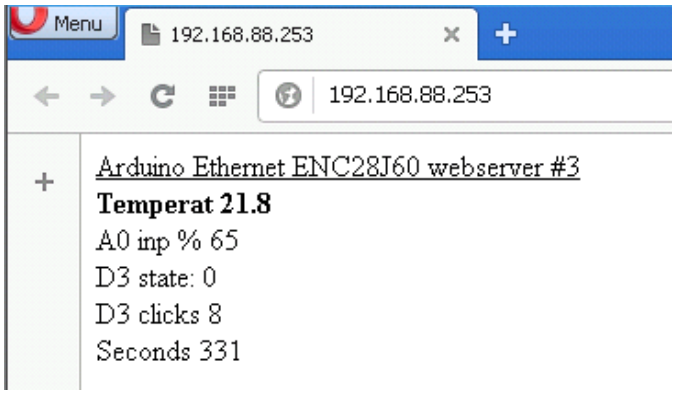
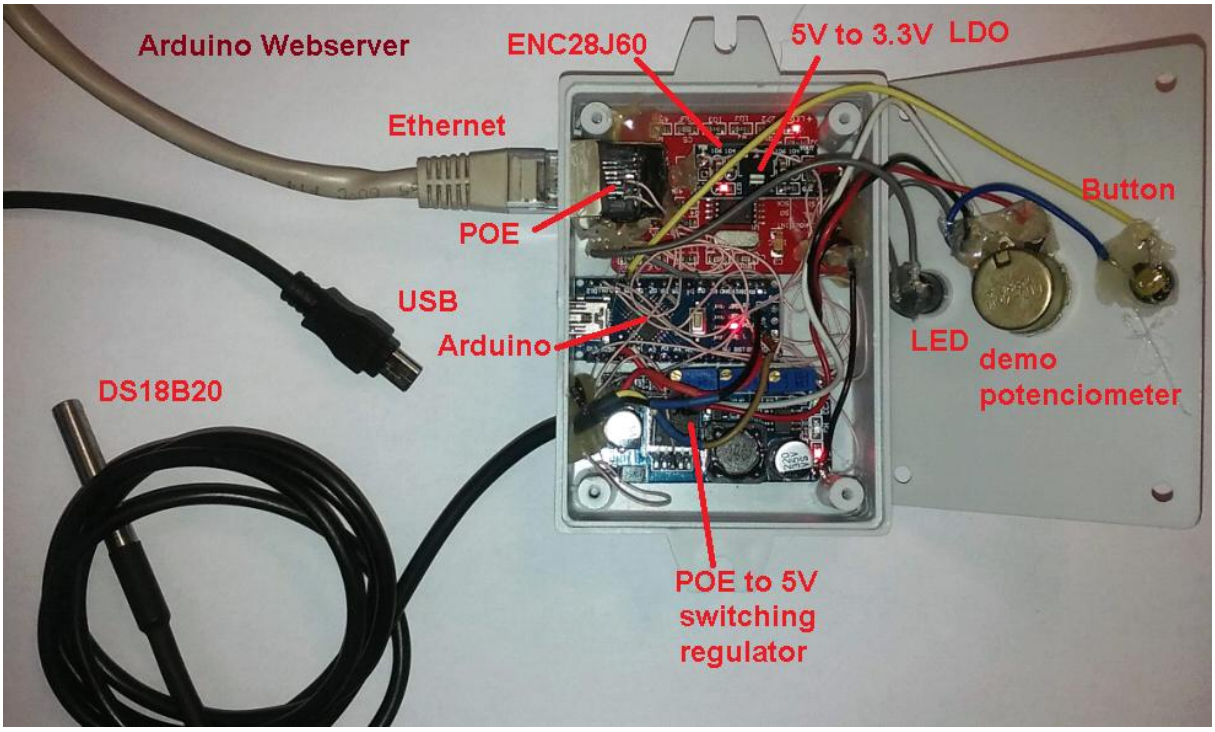
Arduino Ethernet Webservers ENC28J60 with thermometer DS18B20 and Power Over Ethernet POE for ZABBIX IOT data logging

Low-cost Arduino Ethernet webservers are reliable and attractive for wide range of sensor monitoring: temperature, voltage or switches. Ethernet is older than WiFi but more reliable and is widely used in industry. To make such a weberver parts cost ca 10 EUR and it takes one day to build. Commercial Arduino Ethernet thermometers are available on Ebay starting at 50 EUR.

Arduino webserver is powered via 5V USB connector or with more efforts Power Over Ethernet (POE) capability is added. Arduino Nano can be reprogrammed any time via the USB connector.





Free-of-charge ZABBIX IOT platform for plotting and monitoring data

ZABBIX is a platform for IOT signal monitoring developed for server room monitoring and for sending out alarms. It is similar to Xively, but ZABBIX provides installation download free of charge and can be installed also on Raspberry Pi. The company's business model is to send a service person to install Zabbix in industrial client server rooms. ZABBIX home is Riga the capital city of Latvia, EU. Zabbix has been actively developed during the last decade:

http://www.zabbix.com

ZABBIX can be installed on a Linux PC or on a virtual machine. It works with a SQL database. A virtual machine image can also be downloaded. Configuration is done via a webpage.

Large flexibility is achieved that computers to be monitored just need to generate a webpage and ZABBIX just fetches data from a webpage of other servers. This allows to monitor web traffic, server load and if server is alive.

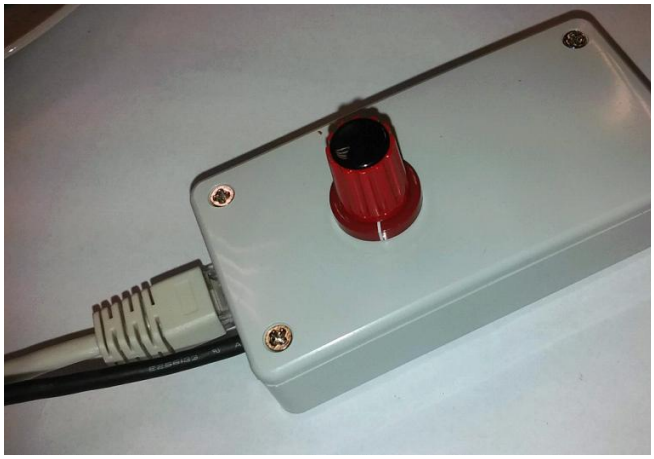
In recent years has opened a possibility to make low-cost Arduino Ethernet webservers for electric sensor monitoring: temperature, voltage or switches. Here is the vision by ZABBIX:

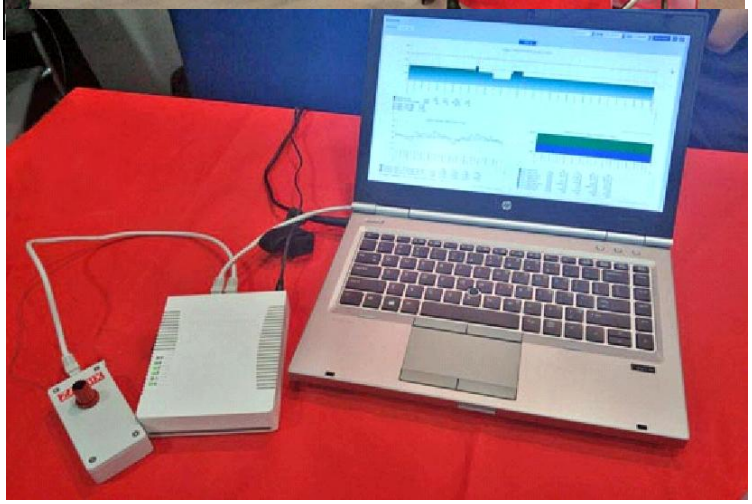
Envisioning picture made by Zabbix for DevTernity conference 12.2016. In Riga, Latvia.

ZABBIX stand at a developer conference

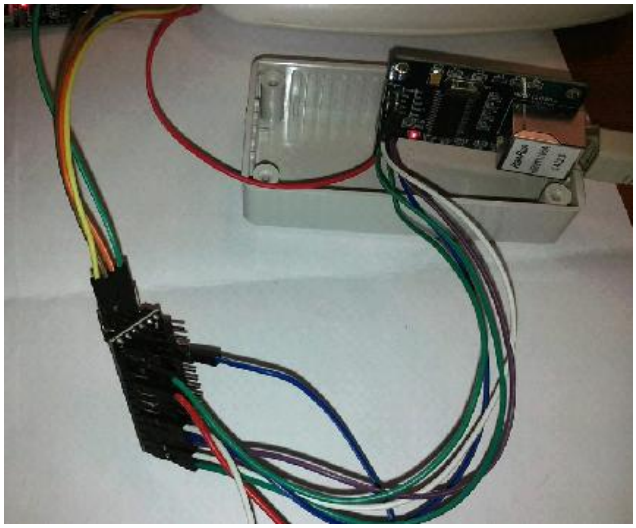
Demo Arduino Ethernet webserver was built for the DevTernity conference and exhibition 2016.12. 01 in Riga, Latvia.

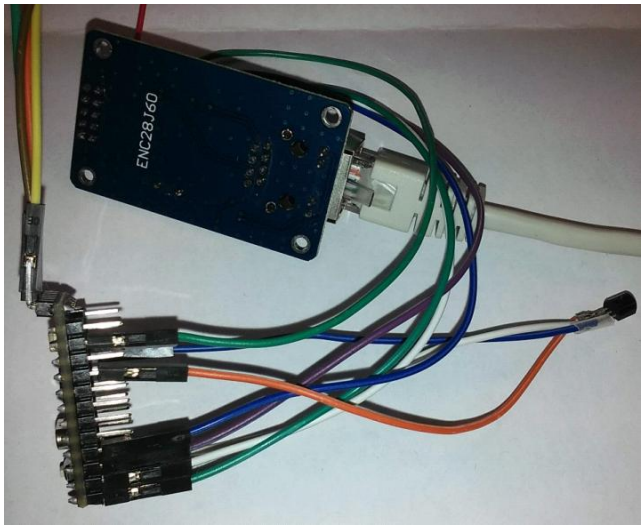
Potentiometer knob simulated filling a tank and real-time graph appeared online in Zabbix.





Inside the box was packed with this. Version on the right had a DS18B20 temperature sensor.

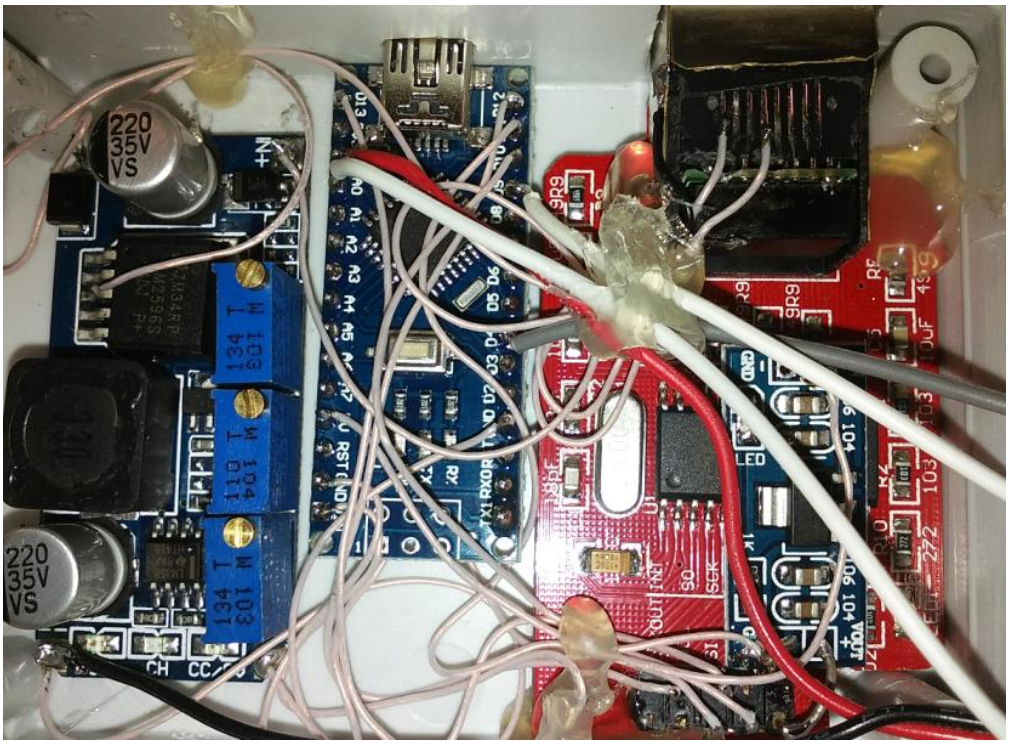




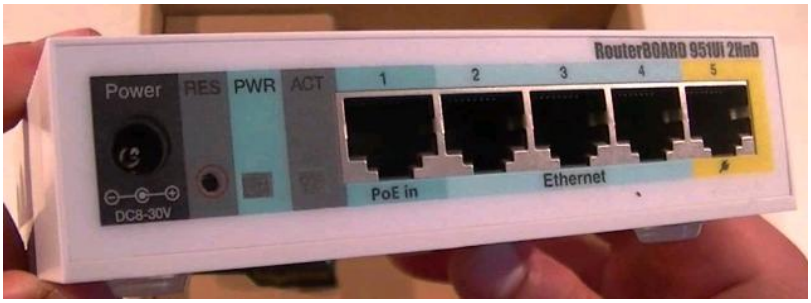
Power over Ethernet to supply the webserver

Power over Ethernet was a tricky thing to achieve. Ethernet connector of ENC28J60 shield has a transformer inside that galvanically separates inputs from outputs. To get POE the metal shield of the connector was cut off with a diamond disk allowing to access connections before the transformer. The connector pins had to be cut off from the circuit board with a transformer because there was a low-resistance connection. Thin flexible wires were soldered to POE pins.

A correct solution would be to use Ethernet connector without the built-in transformer, found in routers or a-10pin Ethernet connector, giving access to POE pins, Farnell ca 8EUR.



As a source of POE *Mikrotik routerboard 951Ui 2HnD* was used. It has 5 Ethernet ports. Nr 1 is marked POE input. There is no voltage across it. Nr 5 is marked POE output and to activate the POE output voltage one needs to change Configuration/Interfaces/POE /ON. Routerboard is supplied from 24 V 2A power supply. This 24V voltage appears as POE output voltage. There are other routers providing more POE outputs.



Tweaking a standard Ethernet connector for POE use

1) Lift up the metal cover of the Eth connector.

2) Cut with a knife the plastic cover.

3) Identify the contacts and cut off using a 1 mm or 1.5 mm diameter mill (Dremel). This is necessary because pins are soldered internal resistors that block POE activation from router side.

4) Using a thin soldering iron tip solder two wires to the two contacts.

5) Put some insulation tape and close the metal cover of the connector. Fix the new POE wires with some glue.

6) To ensure correct POE polarity add a rectifier diode bridge before the step-down regulator.

7) Step-down module is LM2596HV module can be used. Industrial POE modules use a galvanic isolation transformer. It might be possible to use a phone charger from 110V to 5V as it might work at lower voltage.

http://www.tuxgraphics.org/electronics/200903/hobby-poe.shtml

In POE Ethernet specs:
4 5 is + orange wires
7 8 is - blue wires

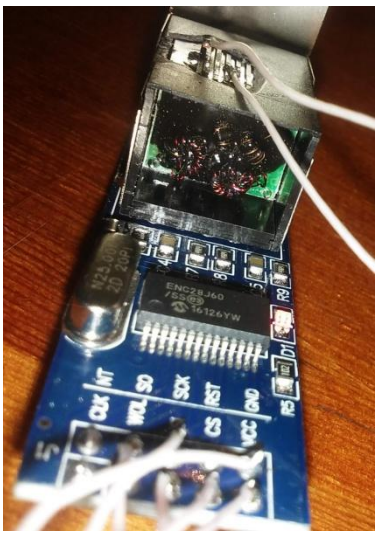


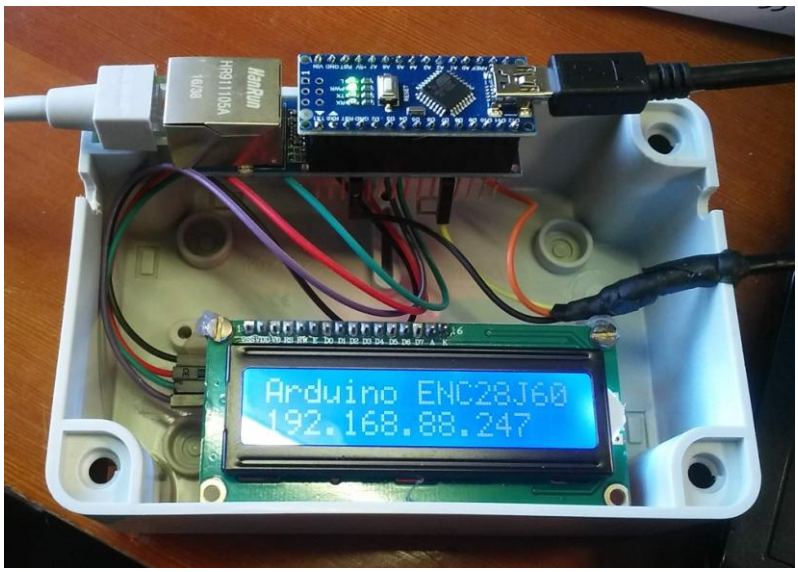
Photo on the right shows the Arduino server box running on POE with USB cable unplugged.

LCD and box with transparent lid

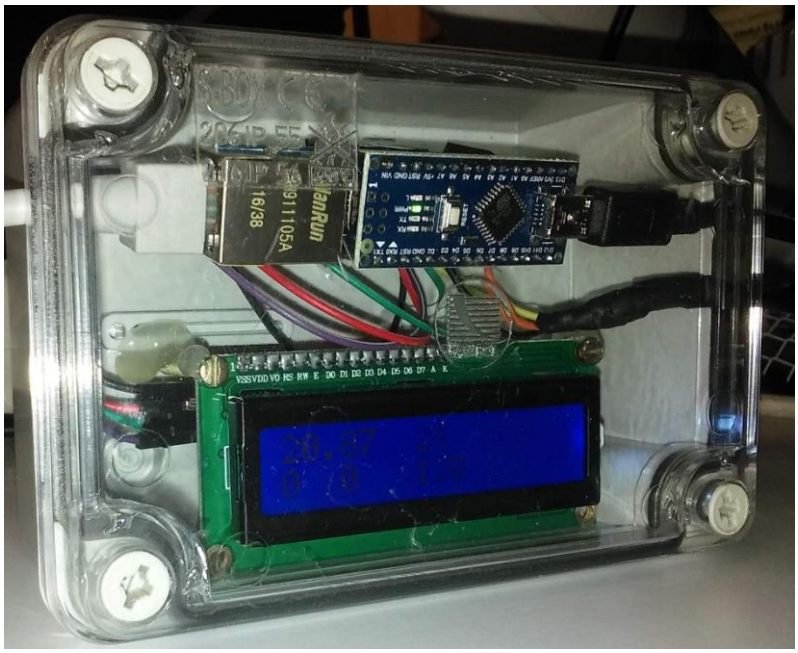
LCD indicator is a very handy thing to see the obtained IP address on startup and allows easy to check that the device is working correctly. The two-line LCD indicator has an adapter board with I2C-expander PCF8574 that only needs 2 wires (data and clock). One needs to identify the I2C address of this chip first as it might differ from different suppliers. Two-line indicator is bulky and next version will be with a tiny OLED graphical indicator, but OLEDs might not survive years of continuous operation.

I2C LCD screen 2-line PCF8574T connects to Arduino A4=SCL A5= SDA and uses the library: https://bitbucket.org/fmalpartida/new-liquidcrystal/downloads

ENC board from Ebay was used that into it Arduino nano. It is more expensive, but has 3.3V regulator chip. It saves work on soldering wires, but USB connector is on another side that is somewhat ugly. There is no POE in this version.

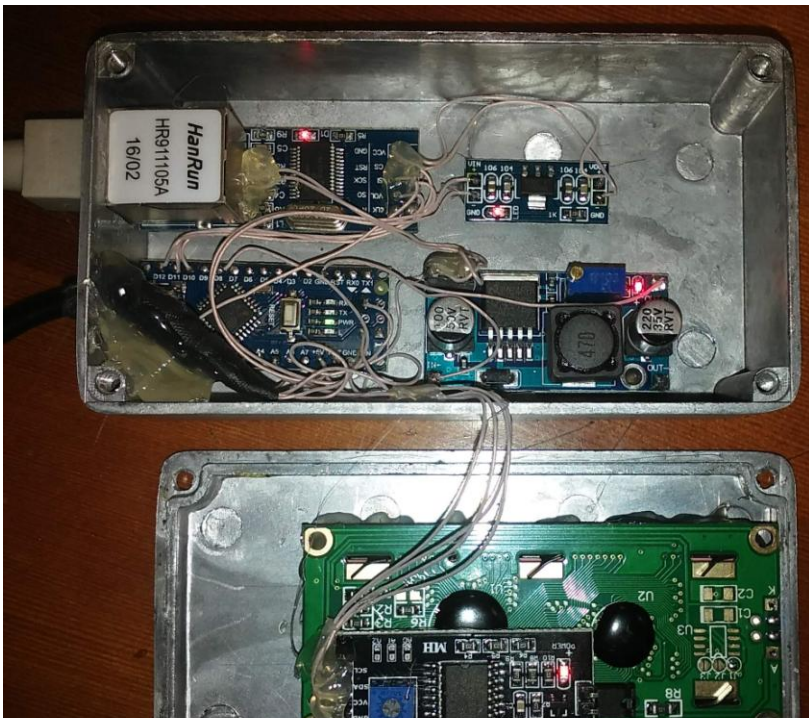


This version has the quickest and easiest assembly of all. One needs to file holes for cables and to glue LCD. That's it. But in order to change Eth cable or USB cable one needs to unscrew the box lid which is an easy work. Nice dust protection.



Metal enclosure and LCD

For professional look a metal box is used as enclosure, for example, “guitar-pedal box” from Ebay. Necessary holes for Ethernet, USB and LCD are cut using a drill, mill and filing. Circuit boards are fixed into the metal enclosure using hot glue. It might not be the professional solution, but good for starting.



Aduino IDE

Program initially was written in Arduino IDE 1.0.0. Later it was decided to switch to one of the latest Arduino IDE 1.6.13. Differences were in OneWire library use. I use D8 for the one-wire bus as D9 can be useful for PWM output. Quick sensor fidelity test can be done measuring human body temperature.

Ethernet library for ENC28J60 allows to choose between static IP or DHCP addresses, but Mac address needs to be defined manually unique for every new box on the same network: https://github.com/jcw/ethercard

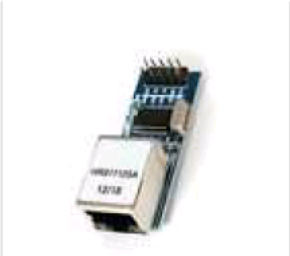
ENC SCK -> Arduino pin 13
ENC SO -> Arduino pin 12
ENC SI -> Arduino pin 11
ENC CS -> Arduino pin 10 (some other online examples use pin 8)
ENC VCC -> 3V3 from USB serial adapter CP2102 or PL2303 but not enough from CH340
ENC GND -> GND

Shopping on Ebay

Arduino could come without or with soldered connectors and USB cable included or not.

MINI USB Nano V3.0
ATmega328P CH340G 5V
16M Micro-controller board For
Arduino N
(141975930194)

There are two variations of ENC28J60 board with wire-jumper connections.



1PCS MiNi ENC28J60
Ethernet LAN Network Module
For Arduino SPI AVR PIC LPC
(381447240602)

To make 3.3V out of 5V for the ENC board is not enough current from Arduino CH340 chip.

5Pcs DC/DC 4.5V-7V to 3.3V AMS117-3.3V Power Supply
Module Voltage Regulator
(222047956286)

Another ENC28J60 board modificationis more expensive with plug in socket for Arduino Nano v3. It has a 3.3V regulator on the board and also TTL level shifter.

Mini ENC28J60 Webserver module Ethernet Shield board for
Arduino Nano v3.0 top
(221629702393)

If you decide to go for POE then you need a step-down POE voltage regulator that can handlle up to 57 Volts, for example LM2576 HV version

LM2576HV DC Step Down 5V
60V~1.25V-30V Adjustable
Power Supply Modul
(262720288899)

Some routers and switches can supply POE, if you don't have such model then can inject POE.

POE Module Injector Over Ethernet Router 4 LAN + Power port for IP Camera 48V
(361261192568)

Passive Power Over Ethernet PoE Adapter Injector Splitter Kit 5v 12v 24v 48v
(141919085135)

Industrial practice is to us metal enclosures. Simples but not nicest is

1590B Style Aluminum Stomp Box Effects Pedal Enclosure FOR Guitar Hotsell HT
(201501892493)

Hermetically sealed sensors with cable are convenient to use. There are some with audio plug on the end.

5PCS Waterproof Digital Thermal Probe or Sensor DS18B20
(140759967888)

New Blue IIC I2C TWI 1602 16x2 Serial LCD Module Display for Arduino
(221439853893)

0.96" I2C IIC SPI Serial 128X64 White OLED LCD LED Display Module for Arduino
(172241275240)

5V 2A USB EU Plug Wall Charger Fast Charging Home Travel Adjustor Power Adaptor
(162174107551)

Arduino IDE 1.0.1 code for version with a temperature sensor and analog input, no LCD:

```
// Arduino Webserver with Ethernet shield ENC28J60. Static IP or DHCP.
// Webpage shows temperature from sensor DS18B20 and time since reset.
// Library https://github.com/jcw/ethercard ethercard-master rename to ethercardmaster
// 2010-05-28 <jc@wippler.nl> http://opensource.org/licenses/mit-license.php
// Library https://github.com/milesburton/Arduino-Temperature-Control-Library rename to DallasTemperature
// Dallas library is buggy. Examples compiled often show -127.
// Code adopted by Janis Alnis 2016.11.30. Compiled with Arduino IDE version 1.0.0.

// ENC SCK -> Arduino pin 13
// ENC SO  -> Arduino pin 12
// ENC SI  -> Arduino pin 11
// ENC CS  -> Arduino pin 10  (some other online examples use pin 8)
// ENC VCC ->  3V3 from USB serial adapter CP2102 or PL2303 but not enough from CH340
// ENC GND -> GND

#include <EtherCard.h>
// ethernet interface mac address, must be unique on the LAN
static byte mymac[] = { 0x74,0x69,0x69,0x2D,0x30,0x33 };

#define STATIC 0  //  DHCP=0 ,  static=1
#if STATIC
static byte myip[] = { 192,168,1,200 }; // static ip address
#endif

byte Ethernet::buffer[500]; BufferFiller bfill;


#include <OneWire.h>
#include <DallasTemperature.h>
#define ONE_WIRE_BUS 8
OneWire oneWire(ONE_WIRE_BUS); DallasTemperature sensors(&oneWire);
float t1; // temperature
int button=3; long state=0; long clicks=0; // for the digital input button
long AIN;

void setup(){ Serial.begin(9600);
  Serial.println("\nArduino Webserver with ENC28J60 Ethernet");
  Serial.println ("Temperature sensor DS18B20 on D9");
  pinMode(button, INPUT); digitalWrite(button, HIGH); // button
  if (ether.begin(sizeof Ethernet::buffer, mymac) == 0)
    Serial.println( "Failed to access Ethernet controller");
#if STATIC
   ether.staticSetup(myip);
```

```
#else
  if (!ether.dhcpSetup())
    Serial.println("DHCP failed");
#endif
  ether.printIp("IP:  ", ether.myip);
}


static word homePage() {

  long T1=int(t1); long T1A=t1*10-10*T1; // fractional part
  long t = millis() / 1000;

bfill = ether.tcpOffset();
  bfill.emit_p(PSTR(
   "HTTP/1.0 200 OK\r\nContent-Type: text/html\r\nPragma: no-cache\r\n\r\n<meta http-equiv='refresh'
content='1'/>"
   "<body><html><u>Arduino Ethernet ENC28J60 webserver #3</u><BR><B>Temperat $L.$L </B><BR>A0 inp % $L
<br>"
   "D3 state: $L <br>D3 clicks $L <br>Seconds $L </body></html>"),
     T1, T1A, AIN, state, clicks, t);
  return bfill.position();
}


void loop () {
  if (millis()%100>70){if (digitalRead(button)==LOW) {if (state==0) {clicks=clicks+1; state=1;}}
    delay(50); if (digitalRead(button)==HIGH){if (state==1) {state=0;}}}

  if (millis()%1000>900){sensors.requestTemperatures(); t1 = sensors.getTempCByIndex(0);}
//    Serial.print ("t1= ");  Serial.println (t1);}    // t2 = sensors.getTempCByIndex(1);

AIN=analogRead(A0); AIN=AIN*100/1023;   analogWrite(9, AIN);

  word len = ether.packetReceive();  word pos = ether.packetLoop(len);
    if (pos) ether.httpServerReply(homePage()); // if valid tcp data received send web page
}
```

BASH script to read a html webpage, to extract temperature value and to upload it to IOT:

```
# bash script to read the Arduino webserver webpage, extract temperature and upload to Xively.
# webpage: <meta http-equiv='refresh' content='1'/><body><html><h1>Arduino webserver</h1>
# <h1>T= 19.5 </h1> <h1>A0= 25 </h1> <h1>t= 750669</h1>
# a file is prepared in RAM /tmp directory that is uploaded to server.
cd /tmp
rm index*
wget 192.168.1.101
a=$(cat index.html)

# select from webpage the element which is temperature
set -- $a
t=$6
echo $t

# convert scaling
round(){
echo $(printf %.$2f $(echo "scale=$2;(((10^$2)*$1)+0.5)/(10^$2)" | bc))
};


tt=$(round $t/1 0)
echo $tt

if [ $tt -ge 10 ] && [ $tt -lt 50 ]; then

echo $t >/tmp/asi

echo "uploading data to Pachube"

a='{"version":"1.0.0","datastreams":[{"id":"2", "current_value":"'
b='"}]}'
echo $a$t$b
echo $a$t$b > /tmp/update.json

curl --request PUT --header "X-PachubeApiKey: qpLG77lHBQVhhJlJ5yAhOAin_CggAaW5tn
TVnyGj09k" \
--data-binary @/tmp/update.json http://api.pachube.com/v2/feeds/128214.json
Fi
```

Arduino IDE 1.6.13 code for the box with I2C LCD, thermometer, analog input, pushbutton counter:

```
// Arduino Webserver with Ethernet shield ENC28J60. Static IP or DHCP.
// Webpage shows temperature from sensor DS18B20 and time since reset.
// Library https://github.com/jcw/ethercard ethercard-master
// 2010-05-28 <jc@wippler.nl> http://opensource.org/licenses/mit-license.php

// OneWire DS18S20, DS18B20, DS1822 Temperature Example
// http://www.pjrc.com/teensy/td_libs_OneWire.html
// http://milesburton.com/Dallas_Temperature_Control_Library

// I2C LCD screen 2 line PCF8574T I2C LCD Backpack
//Uses library from https://bitbucket.org/fmalpartida/new-liquidcrystal/downloads
#include <Wire.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C  lcd(0x3F,2,1,0,4,5,6,7); // 0x27 is the I2C bus address, SDA=A4, SCL=A5
// I2C adderss find: http://www.instructables.com/id/I2C-LCD-Controller-the-easy-way/?ALLSTEPS


// Code adopted by Janis Alnis. v5. 2017.01.09. Compiled with Arduino IDE version 1.6.13.

// ENC SCK -> Arduino pin 13
// ENC SO  -> Arduino pin 12
// ENC SI  -> Arduino pin 11
// ENC CS  -> Arduino pin 10  (some other online examples use pin 8)
// ENC VCC ->  3V3 from USB serial adapter CP2102 or PL2303 but not enough from CH340
// ENC GND -> GND

#include <EtherCard.h>
// ethernet interface mac address, must be unique on the LAN
static byte mymac[] = { 0x74,0x69,0x69,0x2D,0x30,0x35 };

#define STATIC 0  //  DHCP=0 ,  static=1
#if STATIC
static byte myip[] = { 192,168,1,200 }; // static ip address
#endif

byte Ethernet::buffer[500]; BufferFiller bfill;


#include <OneWire.h>
OneWire  ds(8);  // on pin 8 (a 4.7K resistor is necessary)
float t1; // temperature
int button=3; long state=0; long clicks=0; // for the digital input button
long AIN;

void setup(){ Serial.begin(9600);

 lcd.begin (16,2); lcd.setBacklightPin(3,POSITIVE);
 lcd.setBacklight(HIGH); lcd.home (); lcd.print("Arduino ENC28J60");

 Serial.println("\nArduino Webserver with ENC28J60 Ethernet");
 Serial.println ("Temperature sensor DS18B20 on D8");
 pinMode(button, INPUT); digitalWrite(button, HIGH); // button
 if (ether.begin(sizeof Ethernet::buffer, mymac) == 0)
  Serial.println( "Failed to access Ethernet controller");
#if STATIC
  ether.staticSetup(myip);
#else
 if (!ether.dhcpSetup())
    Serial.println("DHCP failed");
#endif
 ether.printIp(ether.myip);
 lcd.setCursor(0,1);
 lcd.print(ether.myip[0]); lcd.print("."); lcd.print(ether.myip[1]); lcd.print(".");
 lcd.print(ether.myip[2]); lcd.print("."); lcd.print(ether.myip[3]);
 delay(5000); lcd.clear();
 }
```

```
static word homePage() {
bfill = ether.tcpOffset();
  long T1=abs(int(t1)); long T1A=abs(t1*100)-abs(100*T1); // fractional part
  long t = millis() / 1000;
Serial.println (T1);
Serial.println (T1A);
if (t1>0){bfill.emit_p(PSTR(
    "HTTP/1.0 200 OK\r\nContent-Type: text/html\r\nPragma: no-cache\r\n\r\n<meta http-equiv='refresh' content='1'/>"
    "<body><html><u>Arduino Ethernet ENC28J60 webserver #3</u><BR><B>Temperat $L.$L </B><BR>A0 inp % $L <br>"
    "D3 state: $L <br>D3 clicks $L <br>Seconds $L </body></html>"), T1, T1A, AIN, state, clicks, t); }
if (t1<0){  // need a minus sign in text
  bfill.emit_p(PSTR(   // need a minus sign in text
    "HTTP/1.0 200 OK\r\nContent-Type: text/html\r\nPragma: no-cache\r\n\r\n<meta http-equiv='refresh' content='1'/>"
    "<body><html><u>Arduino Ethernet ENC28J60 webserver #3</u><BR><B>Temperat -$L.$L </B><BR>A0 inp % $L <br>"
    "D3 state: $L <br>D3 clicks $L <br>Seconds $L </body></html>"), T1, T1A, AIN, state, clicks, t); }

  return bfill.position();
}




void loop () {
  if (millis()%100>70){if (digitalRead(button)==LOW) {if (state==0) {clicks=clicks+1; state=1;}}
    delay(50); if (digitalRead(button)==HIGH){if (state==1) {state=0;}}}

  if (millis()%1000>900){t1 = getTemp();
  lcd.setCursor (0,0); lcd.print(t1);
  lcd.setCursor (7,0); lcd.print(AIN);
  lcd.setCursor (0,1); lcd.print(state);
  lcd.setCursor (3,1); lcd.print(clicks);
  lcd.setCursor (7,1); lcd.print(millis()/1000);

  }

//    Serial.print ("t1= ");  Serial.println (t1);}    // t2 = sensors.getTempCByIndex(1);

AIN=analogRead(A0); AIN=AIN*100/1023;   analogWrite(9, AIN);

  word len = ether.packetReceive();  word pos = ether.packetLoop(len);
    if (pos) ether.httpServerReply(homePage()); // if valid tcp data received send web page
}




float getTemp(void) {
  byte i;
  byte present = 0;
  byte type_s;
  byte data[12];
  byte addr[8];
  float celsius;
    if ( !ds.search(addr)) {
//    Serial.println("No more addresses.");
//    Serial.println();
//    ds.reset_search();
//    delay(250);
//    return;
  }

//  Serial.print("ROM =");
//  for( i = 0; i < 8; i++) {Serial.write(' ');Serial.print(addr[i], HEX);}

//  if (OneWire::crc8(addr, 7) != addr[7]) {Serial.println("CRC is not valid!");return;}
  //Serial.println();
```

```
  // the first ROM byte indicates which chip
  switch (addr[0]) {
    case 0x10:
//      Serial.println("  Chip = DS18S20");  // or old DS1820
      type_s = 1;
      break;
    case 0x28:
//      Serial.println("  Chip = DS18B20");
      type_s = 0;
      break;
    case 0x22:
//      Serial.println("  Chip = DS1822");
      type_s = 0;
      break;
    default:
//      Serial.println("Device is not a DS18x20 family device.");
      return;
  }

  ds.reset();
  ds.select(addr);
  ds.write(0x44, 1);       // start conversion, with parasite power on at the end

  delay(1000);     // maybe 750ms is enough, maybe not
  // we might do a ds.depower() here, but the reset will take care of it.

  present = ds.reset();
  ds.select(addr);
  ds.write(0xBE);         // Read Scratchpad

//  Serial.print("  Data = ");
//  Serial.print(present, HEX);
//  Serial.print(" ");
  for ( i = 0; i < 9; i++) {           // we need 9 bytes
    data[i] = ds.read();
//    Serial.print(data[i], HEX);
//    Serial.print(" ");
  }
//  Serial.print(" CRC=");
//  Serial.print(OneWire::crc8(data, 8), HEX);
//  Serial.println();

  int16_t raw = (data[1] << 8) | data[0];
  if (type_s) {
    raw = raw << 3; // 9 bit resolution default
    if (data[7] == 0x10) {
      // "count remain" gives full 12 bit resolution
      raw = (raw & 0xFFF0) + 12 - data[6];
    }
  } else {
    byte cfg = (data[4] & 0x60);
    // at lower res, the low bits are undefined, so let's zero them
    if (cfg == 0x00) raw = raw & ~7;  // 9 bit resolution, 93.75 ms
    else if (cfg == 0x20) raw = raw & ~3; // 10 bit res, 187.5 ms
    else if (cfg == 0x40) raw = raw & ~1; // 11 bit res, 375 ms
    //// default is 12 bit resolution, 750 ms conversion time
  }
  celsius = (float)raw / 16.0;
//  Serial.print("  Temperature = ");
//  Serial.print(celsius);
//  Serial.print(" Celsius, ");
  return celsius;
  }
```