```
/*

 * ********************************************************
 *
 *              LIBRARY VER 1.4
 *
 * ********************************************************
 * ********************************************************
 * Mods 20140807 JRS for MODIFIED iTead iBoard changes (see Evernote "Cheap
single Board Ethernet Arduino Interface" )
 * Rewired board pins
 * - MOSI
 * - MISO
 * - SCK
 * And code changes for reassigned CE/SPI pins
 * - CE Radio Pin 3
 * - SPI Ss Pin 8
 * And using the WizNet Ethernet interface
 * Prorgam as Arduino "Duemilanove" at 3.3V
20140807 JRS
 * ********************************************************

 * Copyright (C) 2013 Henrik Ekblad <henrik.ekblad@gmail.com>
 *
 * Contribution by a-lurker
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * version 2 as published by the Free Software Foundation.
 *
 * DESCRIPTION
 * The EthernetGateway sends data received from sensors to the ethernet link.
 * The gateway also accepts input on ethernet interface, which is then sent
out to the radio network.
 *
 * The GW code is designed for Arduino 328p / 16MHz.  ATmega168 does not have
enough memory to run this program.
 *
 * COMPILING
 * You must make sure to disable DEBUG in Sensor.h before compiling this
sketch. Othervise the sketch won't fit in program space when downloading.
 * For UIPEthernet(ENC28J60) usage: Note that I had to disable UDP and DHCP
support in uipethernet-conf.h to reduce space. (which meas you ave to choose
a static IP)
 * For WizNET usage: Do *not* install the provided UIPEthernet-library. Remove
UIPEthernet-include below and uncomment the Ethernet.h.
 *
 * VERA CONFIGURATION:
 * Enter "ip-number:port" in the ip-field of the Arduino GW device. This will
temporarily override any serial configuration for the Vera plugin.
 * E.g. If you want to use the defualt values in this sketch enter:
192.168.178.66:5003
 *
 * LED purposes:
 * - RX (green) - blink fast on radio message recieved. In inclusion mode will
blink fast only on presentation recieved
 * - TX (yellow) - blink fast on radio message transmitted. In inclusion mode
will blink slowly
 * - ERR (red) - fast blink on error during transmission error or recieve crc
error
 *
 *  ----------- Connection guide
 ------------------------------------------------------------------------
 *  13  Radio & Ethernet SPI SCK
 *  12  Radio & Ethernet SPI MISO (SO)
 *  11  Radio & Ethernet SPI MOSI (SI)
 *  10  Ethernet SPI Slave Select (CS)    Pin 10, the SS pin, must be an o/p
```

```
 to put SPI in master mode
 *  9   Radio TX LED using on board LED   (optional)  +5V -> LED -> 270-330
 Ohm resistor -> pin 9.
 *  8   Radio RX LED                      (optional)  +5V -> LED -> 270-330
 Ohm resistor -> pin 8.
 *  7   Radio error LED                   (optional)  +5V -> LED -> 270-330
 Ohm resistor -> pin 7.
 *  6   Radio SPI Slave Select
 *  5   Radio Chip Enable
 *  3   Inclusion mode button             (optional), 10K pull down to GND,
 button to VCC)
 *  2   Radio IRQ pin                     (optional), W5100 Int, if linked to
 pin 2)
 *
 ------------------------------------------------------------------------------
 ------------------------------
 * Powering: both NRF24l01 radio and Ethernet(ENC28J60) uses 3.3V
 */

#include <SPI.h>
#include <MySensor.h>
#include <MyGateway.h>
#include <stdarg.h>

// Use this if you have attached a Ethernet ENC28J60 shields
//#include <UIPEthernet.h>

// Use this fo WizNET module and Arduino Ethernet Shield
#include <Ethernet.h>


#define INCLUSION_MODE_TIME 1 // Number of minutes inclusion mode is enabled
#define INCLUSION_MODE_PIN  3 // Digital pin used for inclusion mode button

// #define RADIO_CE_PIN        5  // radio chip enable
#define RADIO_CE_PIN        3  // radio chip enable 20140830
// #define RADIO_SPI_SS_PIN    6  // radio SPI serial select
#define RADIO_SPI_SS_PIN    8  // radio SPI serial select
20140807 JRS

#define RADIO_ERROR_LED_PIN 7  // Error led pin
// #define RADIO_RX_LED_PIN    8  // Receive led pin
20140807 JRS
#define RADIO_RX_LED_PIN    8  // Receive led pin
#define RADIO_TX_LED_PIN    9  // the PCB, on board LED

// #define IP_PORT 5003        // The port you want to open
#define IP_PORT 81        // The port you want to open
20140807 JRS
//IPAddress myIp (192, 168, 178, 66);  // Configure your static ip-address
here
IPAddress myIp (192, 168, 15, 119);  // Configure your static ip-address here
20140807 JRS

// The MAC address can be anything you want but should be unique on your
network.
// Newer boards have a MAC address printed on the underside of the PCB, which
you can (optionally) use.
// Note that most of the Ardunio examples use  "DEAD BEEF FEED" for the MAC
address.
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };  // DEAD BEEF FEED

// a R/W server on the port
EthernetServer server = EthernetServer(IP_PORT);

// No blink or button functionality. Use the vanilla constructor.
MyGateway gw(RADIO_CE_PIN, RADIO_SPI_SS_PIN, INCLUSION_MODE_TIME);
```

```
// Uncomment this constructor if you have leds and include button attached to
your gateway
//MyGateway gw(RADIO_CE_PIN, RADIO_SPI_SS_PIN, INCLUSION_MODE_TIME,
INCLUSION_MODE_PIN, RADIO_RX_LED_PIN, RADIO_TX_LED_PIN, RADIO_ERROR_LED_PIN);


char inputString[MAX_RECEIVE_LENGTH] = "";    // A string to hold incoming
commands from serial/ethernet interface
int inputPos = 0;

void setup()
{
  // Initialize gateway at maximum PA level, channel 70 and callback for write
  operations
  gw.begin(RF24_PA_LEVEL_GW, RF24_CHANNEL, RF24_DATARATE, writeEthernet);

  Ethernet.begin(mac, myIp);

  // give the Ethernet interface a second to initialize
  delay(1000);

  // start listening for clients
  server.begin();
}

// This will be called when data should be written to ethernet
void writeEthernet(char *writeBuffer) {
  server.write(writeBuffer);
}


void loop()
{
  // if an incoming client connects, there will be
  // bytes available to read via the client object
  EthernetClient client = server.available();

  if (client) {
      // if got 1 or more bytes
      if (client.available()) {
         // read the bytes incoming from the client
         char inChar = client.read();

         if (inputPos<MAX_RECEIVE_LENGTH-1) {
           // if newline then command is complete
           if (inChar == '\n') {
              // a command was issued by the client
              // we will now try to send it to the actuator
              inputString[inputPos] = 0;

              // echo the string to the serial port
              Serial.print(inputString);

              gw.parseAndSend(inputString);

              // clear the string:
              inputPos = 0;
           } else {
             // add it to the inputString:
             inputString[inputPos] = inChar;
             inputPos++;
           }
        } else {
           // Incoming message too long. Throw away
           inputPos = 0;
        }
      }
   }
```

```
   gw.processRadioMessage();
}
```