Arduino

One day I ran into this particular Arduino ENC28J60 Ethernet module on eBay for $18. It included an Arduino ENC28J60 Ethernet shield/module **and** an Arduino USB Nano V3.0. Well, that's pretty much for free isn't it? So I could not resist and bought it, fully well knowing that it might not work. It took a little bit of figuring out, but I finally got it to work.

Playing with the Arduino is definitely fun, specially when you start looking into these kind of fun gadgets (snoop around on eBay and Amazon to see what's out there!). If you look at eBay, or for example Amazon, for an Arduino ENC28J60 Ethernet shield, then you'll notice that there are plenty models variations.

In this article we will focus on selecting a good ethernet library for the ENC28J60 and run a "Hello World" example.
The code discussed in this article will probably work for all of them.

The number "**ENC28J60**" actually only refers to a chip developed by Microchip. This chip has 28 pins and contains a complete stand alone Ethernet controller for a 10BASE-T network connection with an SPI interface so microcontrollers like the Arduino can "talk" to it.

10BASE-T is the same connector you'll find on your computer (if it has one) to connect with a wire to a network, where "10" indicates a maximum speed of 10 Mbit/sec. That might sound slow, but if you consider that it's being used by devices like the Arduino, then you can't really expect massive data loads anyway. I've found it to be very responsive.

For those in need of making cables for this, please read our "**How to make your own network cables**" article.

figure 1: 10BASE-T uses RJ45 Connectors

When looking for one, you'll find that there are numerous variations available, and they pretty much all work the same, just the board and pins look different for specific purposes. The only thing that can be tricky is finding the right pins for the right library. Illustration below: Just two of many variations of the ENC28J60 modules.

I found mine at eBay for $18 which came with an Arduino Nano (left in the picture – Nano not displayed), which works just fine with, for example, an Arduino Uno as well. The connectors however are geared towards mounting an Arduino Nano of course.

My main reasons to pick this one (besides being totally unaware, at the time, that this is **not** the same as the Arduino Ethernet shield) were: Price, came with a Nano and size. The module and Nano combined make this thing VERY compact. Including the Arduino Nano (clone) the setup would be app 6.7 cm (~2.6″) long, 1.7 cm (~0.7″) wide, and 1.7 cm (~0.7″) tall – depending on how you use the pins at the bottom and how you mount your Nano of course. You can choose the top connectors or bottom pins (breadboard) while experimenting, but you could consider cutting off the bottom pins for your final product.

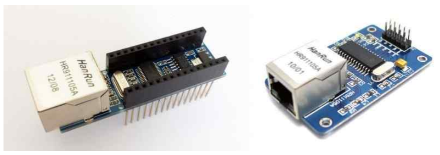

figure 2: ENC28J60 – Two examples of variations

Since I randomly picked this particular model, I had to find out the hard way that things do not seem all that easy, since it's not compatible with the Arduino "Ethernet" library that comes with your Arduino IDE. And then I'm not even mentioning the lack of good info to determine what pins are to be used, so I had to figure that out by myself.

In the next paragraphs you'll find my experiences with three Arduino libraries. **They all work great!**

UIPEthernet is great for projects that require "print" to be fully implemented and need to be a drop-in replacement for the standard "Ethernet" library.
ETHER_28j60 is great for it's simplicity and small size, but comes with limitations.
Ethercard seems best for very advanced users, but I'm sure UIPEthernet can match it's capabilities.

I've included a "Hello World!" example for all three, which could use some optimizing, where the "Hello World!" message can be viewed in your webbrowser.

We will aways need power so we will always need **GND** and **+3.3V** *or* **+5V** pin.
For my eBay module **I had to use the +5V** (it has a voltage regulator onboard to handle that), as the 3.3V pin didn't seem to work.

Below a table, based on a Arduino Uno, Arduino Nano and my eBay Ethernet module, with the needed pins for the three libraries I tested.
As you can see, all of them use the standard SPI pins 10, 11, 12 and 13. Except Ethercard, that seems to use pin 8 for SS, instead of the "standard" pin 10.

table: ENC28J60 Pins and Libraries

Pin name	ETHER_28J60	Ethercard	UIPEthernet	My eBay Module
SS	10	**8** (!)	10	**10**
MOSI (SI)	11	11	11	**11**
MISO (SO)	12	12	12	**12**
SCK	13	13	13	**13**

The Ethernet Controller (ENC28J60) is a so called **SPI device** and uses the SPI pins (10, 11, 12, 13) of your Arduino.

SS stands for **S**lave **S**elect, used to enable or disable the slave device (the Ethernet module in this case).
MOSI stands for **M**aster **O**utput **S**lave **I**nput, or in other words: Arduino OUTPUT (data from Arduino to Ethernet Controller).
MISO stands for the opposite, **M**aster **I**nput **S**lave **O**utput, or: Arduino INPUT (data from Ethernet Controller to Arduino).
SCK is the clock used for SPI timing.

The pins described here will have the "Pin name" usually printed on your Ethernet Module. My eBay module however is fully geared towards the Arduino Nano that came with it and has NO SUCH PIN NAMES for the Ethernet controller, just the pin names of the Arduino pins. Hence the extra column for those who buy the same Ethernet Module, showing the pin number.

To illustrate this, below an illustration of the "**Nano Ethernet Shield**" by "**Deek-Robot**" that I purchased from eBay.

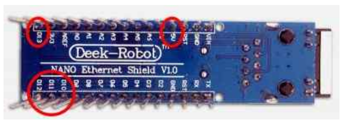

figure 3: My eBay Ethernet module (Deek Robot Nano Ethershield)

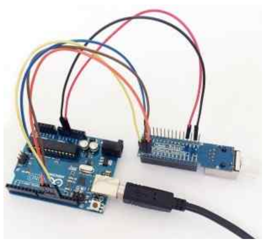

figure 4: My Arduino with ENC28J60 – A wiring example

In this article, I'll only show you a "Hello World!" example for these libraries. In the end "UIPEthernet" became my favorite and for that library I have written other example(s) – I'll add more to this list as they are being published:

- **Data retrieval over network with Data Pull**
- **Data retrieval over network with Data Push**

This is the first library I found, which works great for basic purposes, but I quickly ran into the limitations of the build-in print function. Another problem is that after some digging I found that the development either has stopped development or has been very slow in the past year, which is too bad, because of the (initial) simplicity of this library.

Note that you will need *both* libraries to make this work!

Below the simplicity of ETHER_28J60 … seriously: could it be *any* easier?

The library is also very compact, so it will save memory on your Arduino, compared to the other two libraries.

Downside however is that this library is not compatible with the Ethernet Library code that comes with your Arduino IDE, so it's not a drop-in replacement.

The other downside I ran into was the severely limited "print" function, when it comes to passing for example a String.
My knowledge and experience with normal C-strings is somewhat limited when functions like `sprintf` and `printf` are not there or only partially implemented (limitation of the standard Arduino library).

As development seems to have stopped, I doubt we will see a properly implemented "print" function (unless someone forks it at Github).

ETHER_28J60 and **Ether shield** can be downloaded from Github or you can download it from Tweaking4All.
As always: I recommend getting the latest version from Github, although I have little hope that there will be a newer version in the near future.

Example

```
// A simple web server that always just says "Hello World"

#include "etherShield.h"
#include "ETHER_28J60.h"

// Define MAC address and IP address - both should be unique in your network
static uint8_t mac[6] = {0x54, 0x55, 0x58, 0x10, 0x00, 0x24};
static uint8_t ip[4] = {192, 168, 1, 15};
static uint16_t port = 80; // Use port 80 - the standard for HTTP

ETHER_28J60 ethernet;

void setup()
{
  ethernet.setup(mac, ip, port);
}

void loop()
{
  if (ethernet.serviceRequest())
  {
    ethernet.print("<H1>Hello World</H1>");
    ethernet.respond();
  }
  delay(100);
}
```

This library seems a very well respected in the Arduino community and with good reason. It seems one of the most complete implementations out there.

The code below might look a little bit more complicated, but that's mainly because of the added HTML.

⚠ CAUTION: Ethercard seems to use **pin 8 instead of pin 10**!

Definitely a big plus for this library is that complex tasks like DHCP and such are easy to use, and offers easy accessible advanced features. Definitely excellent for the pro Arduino users.

A big downside (again) is the lack of a simple to use "print" function to sent data, and I'm fully aware that me bitching about it is based on my own limited experience with working with strings and char arrays etc., but I can imagine that I'm not the only one.

Ethercard is, like UIPEthernet, not the smallest library.

EtherCard can be found at GitHub and on their project page or you can download it from Tweaking4All.
Again: I recommend getting the latest and greatest version from Github.

```
#include <EtherCard.h>

// Ethernet IP, default gateway and MAC addresses
static byte myip[] = { 192,168,1,200 };
static byte gwip[] = { 192,168,1,1 };
static byte mymac[] = { 0x74,0x69,0x69,0x2D,0x30,0x31 };

byte Ethernet::buffer[500]; // tcp/ip send and receive buffer

char page[] PROGMEM =
"HTTP/1.0 503 Service Unavailable\r\n"
"Content-Type: text/html\r\n"
"Retry-After: 600\r\n"
"\r\n"
"<html>"
  "<head><title>"
    "Hello World!"
  "</title></head>"
```

Example

```
    "<h3>Hello World! This is your Arduino speaking!</h3>"
  "</body>"
"</html>";

void setup(){
  Serial.begin(57600);
  Serial.println("\n[Hello World]");

  if (ether.begin(sizeof Ethernet::buffer, mymac) == 0)
    Serial.println( "Failed to access Ethernet controller");
  ether.staticSetup(myip, gwip);

  ether.printIp("IP:  ", ether.myip);
  ether.printIp("GW:  ", ether.gwip);
  ether.printIp("DNS: ", ether.dnsip);
}

void loop(){
  // wait for an incoming TCP packet, but ignore its contents
  if (ether.packetLoop(ether.packetReceive())) {
    memcpy_P(ether.tcpOffset(), page, sizeof page);
    ether.httpServerReply(sizeof page - 1);
  }
}
```

After testing the previous two libraries, I ran into UIPEthernet, at this moment my absolute favorite.

You might see the example code below as more complicated, but that is mainly me to blame. I modified and existing example to make a quick "Hello World" for you.

This library is a fully compatible drop-in replacement for the standard Ethernet Library found in your Arduino IDE, which makes it easy to adapt existing examples for use with either the Arduino Ethernet shield for use with the ENC28J60 Ethernet shield. One simply changes the two include lines ("`#include <Ethernet.h>`" and "`#include <SPI.h>`") in standard Ethernet examples to just one include line "`#include <UIPEthernet.h>`".

This library also has a complete implementation of the "print" function that works the same as the "print" function for "Serial", keeping code simple and very easy to use.

Advanced features are available if needed, so "pro" Arduino users might enjoy this library as well.

It will be a little bigger than ETHER_28J60.

UIPEthernet can be found on GitHub, is mentioned on the Arduino website, and optionally you can be downloaded from Tweaking4All.
I recommend getting the latest version from Github.

```
#include <UIPEthernet.h> // Used for Ethernet

// **** ETHERNET SETTING ****
byte mac[] = { 0x54, 0x34, 0x41, 0x30, 0x30, 0x31 };
IPAddress ip(192, 168, 1, 179);
EthernetServer server(80);

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

  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip);
  server.begin();

  Serial.print("IP Address: ");
  Serial.println(Ethernet.localIP());
}

void loop() {
  // listen for incoming clients
  EthernetClient client = server.available();

```

Example

```
{
  Serial.println("-> New Connection");

  // an http request ends with a blank line
  boolean currentLineIsBlank = true;

  while (client.connected())
  {
    if (client.available())
    {
      char c = client.read();

      // if you've gotten to the end of the line (received a newline
      // character) and the line is blank, the http request has ended,
      // so you can send a reply
      if (c == '\n' && currentLineIsBlank)
      {
        client.println("<html><title>Hello World!</title><body><h3>Hello World!</h3></b
        break;
      }

      if (c == '\n') {
        // you're starting a new line
        currentLineIsBlank = true;
      }
      else if (c != '\r')
      {
        // you've gotten a character on the current line
        currentLineIsBlank = false;
      }
    }
  }

  // give the web browser time to receive the data
  delay(10);

  // close the connection:
  client.stop();
  Serial.println("   Disconnected\n");
}
}
```