ZBasic

Application Note

# AN-224 Using Arduino Ethernet Classes in ZBasic

## *Introduction*

This application note describes how to incorporate Arduino Ethernet classes into a ZBasic application. In addition to being useful for adding Ethernet connectivity to an application, the technique can be more generally used for importing other functionalities via C and C++ header files. Note that version v4.1.3 or later of the ZBasic compiler is required in order to successfully compile the code provided with this application note. The code also requires the ZBasic-Arduino compatibility library v1.1 or later (available from the ZBasic website downloads page).

In this application note, four different applications are described illustrating how to use various parts of the Arduino Ethernet classes: an HTTP server, an HTTP client, a UDP server and a UDP (NTP) client.

## *Overview of Arduino Ethernet Classes*

The Arduino Ethernet "library" comprises several C++ classes that implement socket-oriented Ethernet connectivity using the WIZnet W5100 chip, commonly available on Arduino-compatible Ethernet "shields" as well as more generic Ethernet modules, both available from a variety of sources. The W5100 chip incorporates a feature-rich hardwired "Ethernet Stack", supporting the TCP, UDP, IPv4, ICMP, ARP, IGMP and PPPoE protocols. Although the chip operates on 3.3 volts, the SPI interface is 5-volt compatible facilitating simple connections to most microcontrollers. Importantly, the W5100 incorporates a 16K byte internal buffer that can be allocated between up to four concurrent Ethernet connections thus significantly reducing the amount of RAM required on the microcontroller itself.

The open-source Arduino Ethernet Library provides the following classes:

| **Class Name** | **Description** |
|---|---|
| `EthernetClass` | Provides the fundamental infrastructure for a connection to another Ethernet-equipped computer, either as a client or as a server. |
| `EthernetClient` | Provides client-side infrastructure. |
| `EthernetServer` | Provides server-side infrastructure. |
| `EthernetUdp` | Provides the means to send and receive UDP packets. |
| `DNSClient` | Provides infrastructure needed by a DNS client. |
| `DhcpClass` | Provides infrastructure required to obtain an IP address from a DHCP server. |

For applications that do not need either DNS or DHCP services, the Arduino classes have been augmented, adding the following substitute classes that provide the basic functionality with smaller code size:

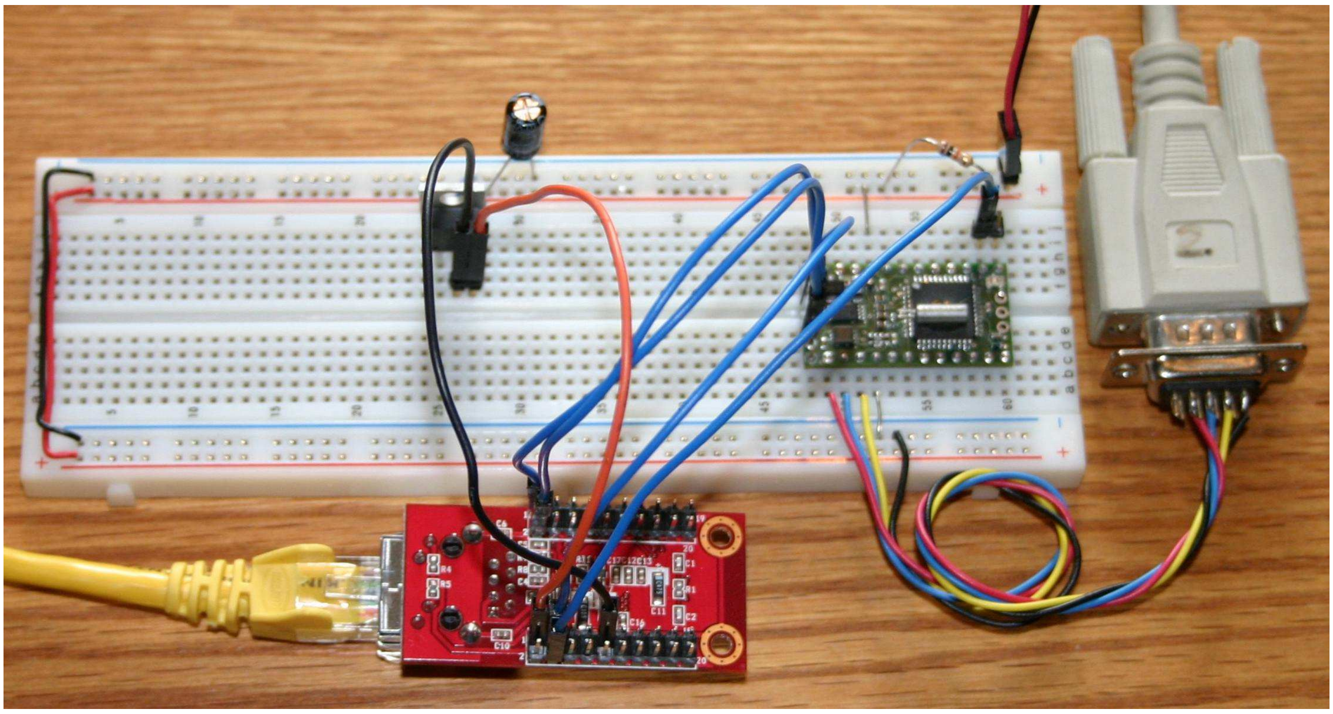
```
EthernetBasicClass
EthernetBasicClient
EthernetBasicServer
EthernetBasicUdp
```

The four sample applications are configurable as to whether they acquire an IP address from a local DHCPserver or, alternatively, use a hard-wired IP address. The two client applications are further configurable as to whether they use DNS to resolve a server name to its IP address or, alternatively, use a hard-wired IP address.

Published February 2013

## *Overview of The Ethernet Hardware*

The photo below shows a simple prototype with the WIZNET W5100 module (available from SparkFun and elsewhere for about $25) connect to a ZX-24a. The module communicates using the SPI interface (blue wires) and requires a 3.3V power supply (red and black wires). In this prototype, the 3.3V supply was derived from the 5V supply using a 3-pin regulator. The SPI slave select line is connected to pin 14 of the ZX-24n.



The W5100 is also available in the form of an Arduino Ethernet "shield" (about $45 from AdaFruit Industries and elsewhere).

## *HTTP Server Application*

Two key elements of all of the Ethernet applications discussed in this application note are illustrated by example below; in this case, the code was excerpted from the file `WebServer.bas`.

```
' specify Arduino compatibility
Option Arduino "C:/Projects/Arduino"

' import the necessary Arduino Ethernet declarations
#import "WebServerImport.h" "#WEB_SERVER_DHCP"
```

The first element is the directive telling the ZBasic compiler to operate in Arduino compatibility mode. The effects of this directive are as follows:

- Enables ZBasic Object Extensions, required in order to access the Arduino class elements.
- Enables C++ code generation, required in order to access the Arduino class elements.
- Specifies the directory in which to find the Arduino header files.
- Specifies the object library containing the ZBasic-Arduino compatibility code.

The second element specifies a header file from which to import definitions. In this particular case, the header file was created specifically for this application and serves as a convenient vehicle for importing the correct set of Arduino header files depending on the desired configuration. The content of the header file `WebServerImport.h` is shown below.

```
#if !defined(WEBSERVER_H_INCLUDED)
#define WEBSERVER_H_INCLUDED

// uncomment the following line to utilize DHCP
//#define WEB_SERVER_DHCP

   // these Arduino Ethernet header files must be imported
   #include "w5100.h"
#if defined(WEB_SERVER_DHCP)
   #include "Ethernet.h"
#else
   #include "EthernetBasic.h"
#endif

#endif // !defined(WEBSERVER_H_INCLUDED)
```

The first two lines together with the last line form what is known as an "include guard". This idiom is commonly used in C and C++ code to prevent the content of an include file from being processed more than once. The fifth line (shown commented out) is used to control which Arduino header files are included further below, making them part of the import process. This is how you select whether to build the application to get its IP address via DHCP or to use a hard-coded IP address.

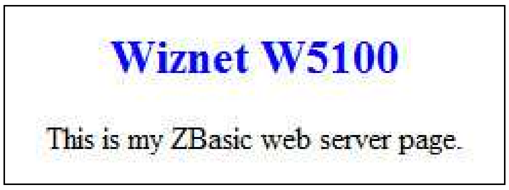
When building the application to use a hard-coded IP address, that IP address and the gateway IP for the local network are specified using the Arduino class `IPAddress` as shown in the excerpt below from `WebServer.bas`; in this case, the host IP address is being set to 192.168.0.31 and the gateway address is being set to 192.168.0.2. Before building the application you'll need to change these IP addresses to be compatible with your own local network.

```
#if Not defined(WEB_SERVER_DHCP)
   ' when DHCP is not used, specify the host and gateway IP addresses
   Private ipAddr as IPAddress = _Create(192, 168, 0, 31)
   Private gwAddr as IPAddress = _Create(192, 168, 0, 2)
#endif
```

One other aspect of the application that must be configured is setting the Media Access Control Address (MAC Address) of the Ethernet Interface. This address comprises a six-byte value that must be unique on each local area network. Because of the length of the address, you could choose a random combination of byte values and have a good chance that it does not match the MAC address of any other device on your local network. To be more confident, you could use the MAC address of a network device that you have on hand, one that is not likely to be connected to the same network. The MAC address of network devices is usually printed somewhere on the device. The MAC address in the example below is from an older, unused Network Interface Card in the author's possession so it should be safe to use in virtually any local area network.

```
' specify the MAC address of the W5100 chip (must be unique on a local network)
Private myMacAddr as ByteVectorData ({ &H00, &H10, &H5a, &H63, &H2e, &H85 })
```

Assuming that you have successfully compiled the server application (configured with a fixed IP address of 192.168.0.31) and downloaded it to the ZBasic device, you should be able to demonstrate that your server is active by pointing your browser to the URL http://192.168.0.31. The resulting browser display should be something like this:

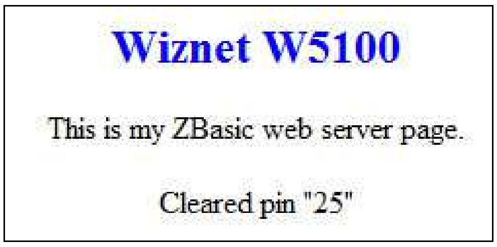

Wiznet W5100

This is my ZBasic web server page.

If the code was compiled with the identifier `QUERY_STRING` defined, the web server supports some simple dynamic content as shown in the table below.

| **Name/Value Pair** | **Description** |
|---|---|
| `getpin=<pin>` | Read the specified pin, output the value. |
| `clrpin=<pin>` | Set the pin to logic zero, output the action taken. |
| `setpin=<pin>` | Set the pin to logic one, output the action taken. |

If you use the URL `http://192.168.0.31?clrpin=25`, the green LED on the ZX-24n should illuminate and the browser should display the following:



Wiznet W5100

This is my ZBasic web server page.

Cleared pin "25"

Beginning with this simple sever application, you should be able to add functionality to suit your needs.

## *HTTP Client Application*

Building on the basic concepts introduced in the HTTP server application, we now demonstrate how to implement an HTTP client application. In this scenario, the code running on the ZBasic device will send a request to a remote server and process the returned information. The importing of Arduino class information is handle the same way as for the server application but using a different intermediate header file named `WebClientImport.h`.

```
' import the necessary Arduino Ethernet declarations
#import "WebClientImport.h" "#WEB_CLIENT_DNS", "#WEB_CLIENT_DHCP"
```

The content of the `WebClientImport.h` header file (exclusive of the include guard) is as follows:

```
// uncomment the following line to utilize DHCP
//#define WEB_CLIENT_DHCP

// uncomment the following line to utilize DNS to resolve names to IP addresses
#define WEB_CLIENT_DNS

   // these Arduino Ethernet header files must be imported
   #include "w5100.h"
#if defined(WEB_CLIENT_DNS)
   #include "Dns.h"
#endif
#if defined(WEB_CLIENT_DHCP)
   #include "Ethernet.h"
   #include "EthernetClient.h"
#else
   #include "EthernetBasic.h"
   #include "EthernetBasicClient.h"
#endif
```

In this case, we have two `#define` identifiers controlling what is imported. The identifier `WEB_CLIENT_DHCP` serves the same purpose as did `WEB_SERVER_DHCP` in the previous example - controlling the assignment of the host IP address. The added identifier, `WEB_CLIENT_DNS`, controls whether we can use a server name in the connection request or if we must use the server's IP address. Being able to connect to a server using its name is much more convenient, of course, but that capability comes at the cost of larger code size.

The code in `WebClient.bas` uses the existence of the identifier `WEB_CLIENT_DNS` to control how the server is addressed by way of the following code excerpt:

```
' specify the name or IP address of the web server
#if defined(WEB_CLIENT_DNS)
   Private Const serverName as String = "www.zbasic.net"
#else
   Private serverIP as IPAddress = _Create(69, 89, 26, 47) ' zbasic.net server
#endif
```

This application is written to allow DNS resolution even if DHCP is not used to obtain an IP address for the client. In the absence of DHCP, the host IP address and the gateway IP address must be specified. Further, if DNS resolution is desired it is also necessary to specify a DNS server IP address to use as illustrated in the following excerpt. In this particular example, the gateway also serves as the DNS server but that is not universally the case.

```
#if Not defined(WEB_CLIENT_DHCP)
   ' specify the host IP address and gateway IP address
   Dim ipAddr as IPAddress = _Create(192, 168, 0, 31)
   Dim gwAddr as IPAddress = _Create(192, 168, 0, 2)
  #if Defined(WEB_CLIENT_DNS)
   ' specify the DNS server IP address
   Dim dnsAddr as IPAddress =  _Create(192, 168, 0, 2)
  #endif
#endif
```

For the purposes of this example, we provide a simple web page on the zbasic.net server that returns HTML code like the following. You can verify this by pointing your browser to `http://www.zbasic.net/EtherTest.php`. The numeric value following the left parenthesis is the current date and time (GMT) in Unix format, i.e., the number of seconds since 01 Jan 1970. Following that is the date and time in the more familiar form.

```
<html><body>
time(1360270595 2013-02-07 20:56:35)<br>
</body></html>
```

You'll find the actual HTTP request in the sample code, appearing as the following excerpt:

```
' specify the web page to load
Private Const requestStr as String = "GET /EtherTest.php HTTP/1.1" & Chr(&H0a)
```

The client application processes the HTML code returned by the server, extracting the date/time information and using it to set the ZBasic RTC. If you compile and download the sample client application you should see output similar to the following.

```
ZBasic v4.1.2
IP address: 192.168.0.31
time set to 07 Feb 2013 20:56:35
```

Using the sample HTTP client application as a starting point, you should be able to change the server and request, add additional servers and requests, etc.

## *UDP Server Application*

The two previous examples involved the use of the HTTP protocol which is built on top of the Transmission Control Protocol (TCP). TCP is a connection-oriented protocol that implements "stream" characteristics (i.e. packets are sent and are guaranteed to arrive in a specific order) and some reliability measures (e.g. automatic re-transmission). While these characteristics are useful, they come at the price of complexity. In contrast, the User Datagram Protocol (UDP) is a simpler, connectionless protocol that is useful in a variety of situations.

In this example application we implement a UDP server that implements the ability to read a pin and to set the state of a pin (similar to the query string capabilities of the HTTP server application). The request messages supported by the server and the response message from the server for each request is shown in the table below.

| **Request** | **Description** |
|---|---|
| `R<pin>` | Read the specified pin, send the value as `0` or `1`. If the pin is missing or invalid, send `nak`. |
| `W<pin>,<val>` | If present and valid, set the pin to the specified state, send `ack`. If the pin is missing or invalid or if the value is missing or invalid, send `nak`. |

As before, the importing of Arduino class elements is controlled by the intermediary header file `UDPServerImport.h`., the content of which (exclusive of the include guard) is shown below.

```
// uncomment the following line to utilize DHCP
//#define UDP_SERVER_DHCP

   // these Arduino Ethernet header files must be imported
   #include "w5100.h"
#if defined(UDP_SERVER_DHCP)
   #include "Ethernet.h"
   #include "EthernetUDP.h"
#else
   #include "EthernetBasic.h"
   #include "EthernetBasicUDP.h"
#endif
```

As before, the UDP server needs an IP address which can either be hard-wired or acquired via DHCP. For demonstration purposes, we will assume that the IP address is 192.168.0.31. Further, a UDP server listens on a particular "port". In this example, we assume that it listens on port 8888.

Once the UDP server application is compiled and downloaded, you'll need a UDP client application in order to test it. One example of such a UDP client is UDPSZ by Luigi Auriemma (freely downloadable from the Internet). Using this simple but powerful UDP client we can demonstrate the functionality of the sample ZBasic UDP server application.

With the udpsz.exe application available, we can issue the following command:

```
udpsz -D -c "W25,0" 192.168.0.31 8888 5
```

The value 5 at the end of the command indicates the length of the message being sent (`W25,0`). Between the message and the length are the server IP address and the port number. The -D option tells udpsz to decode the value returned by the server in response to the request. Running this command should result in the green LED on the ZX-24n illuminating and udpsz reporting that the returned message was `ack`.

Similarly, if a pullup resistor is connected to pin 5, the command below should result in udpsz reporting that the returned message was `1`.

```
udpsz -D -c "R5" 192.168.0.31 8888 2
```

Using this simple example as a starting point, you should be able to expand it to support other messages as necessary.

## *UDP Client Application*

To round out the Ethernet examples, we now present a UDP client application. This particular example uses UDP to communicate with a Network Time Protocol (NTP) server to get the current time and set the ZBasic RTC. As with the other Ethernet examples, we use an intermediary header file, `NetTimeImport.h`, to control importing Arduino classes, the content of which (exclusive of the include guard) is shown below.

```
// uncomment the following line to utilize DHCP
//#define NET_TIME_DHCP

// uncomment the following line to utilize DNS to resolve names to IP addresses
#define NET_TIME_DNS

   // these Arduino Ethernet header files must be imported
   #include "w5100.h"
#if defined(NET_TIME_DNS)
   #include "Dns.h"
#endif
#if defined(NET_TIME_DHCP)
   #include "Ethernet.h"
   #include "EthernetUDP.h"
#else
   #include "EthernetBasic.h"
   #include "EthernetBasicUDP.h"
#endif
```

As with the HTTP client example, if DHCP services are not used and DNS resolution is desired, the IP address of a DNS server must be provided. The code for doing so is functionally identical to that in the HTTP client.

The code excerpt below shows how the address of the NTP server is specified. Note, particularly, the option of specifying an NTP server on your local network. Depending on your network structure, you may or may not have a local NTP server available.

```
#if defined(NET_TIME_DNS)
   Private Const timeServer as String = "us.pool.ntp.org"
#elseif defined(USE_LOCAL_NTP_SERVER)
   Private timeServer as IPAddress = _Create(192, 168, 0, 1)
#else
   Private timeServer as IPAddress = _Create(69, 167, 160, 102) ' us.pool.ntp.org
#endif
```

Compiling and downloading the code should produce output similar to that shown below.

```
ZBasic v4.1.2
IP address: 192.168.0.31
time set to 08 Feb 2013 13:59:49
```

## *Summary*

These four applications demonstrate the fundamentals of importing Arduino Ethernet classes (and Arduino code generally) and illustrate basic Ethernet connections from both client and server perspective. When compiling these applications you may notice the warning indicating that the ZBasic compiler cannot compute stack use. This is due to the use of virtual methods in some of the Arduino classes that are imported. Because of this, if your application utilizes multiple tasks, you'll have to empirically determine the stack size needed for each task.

## *Software*

The ZBasic and associated header files are provided in an accompanying .zip archive.

## *Author*

Don Kinzer is the founder and CEO of Elba Corporation. He has extensive experience in both hardware and software aspects of microprocessors, microcontrollers and general-purpose computers. Don can be contacted via email at dkinzer@zbasic.net.