User Manual

Ethernet API

86Duino Coding IDE 500+

Ethernet Library

(Version1.0)

REVISION

DATE	VERSION	DESCRIPTION
2025/9/10	**Version1.0**	**New Release.**

COPYRIGHT

Ver.1.0 September, 2025

TRADEMARKS ACKNOWLEDGMENT

For more detailed information or if you are interested in other ICOP products, please visit our official websites at:

- Global: www.icop.com.tw
- USA: www.icoptech.com
- Japan: www.icop.co.jp
- Europe: www.icoptech.eu
- China: www.icop.com.cn

For technical support or drivers download, please visit our websites at:

- https://www.icop.com.tw/resource_entrance

For EtherCAT solution service, support or tutorials, 86Duino Coding IDE 500+ introduction, functions, languages, libraries, etc. Please visit the QEC website:

- QEC: https://www.qec.tw/

This Manual is for the QEC series.

SAFETY INFORMATION

- Read these safety instructions carefully.
- Please carry the unit with both hands and handle it with caution.
- Power Input voltage +19 to +50VDC Power Input (Typ. +24VDC)
- Make sure the voltage of the power source is appropriate before connecting the equipment to the power outlet.
- To prevent the QEC device from shock or fire hazards, please keep it dry and away from water and humidity.
- Operating temperature between -20 to +70°C/-40 to +85°C (Option).
- When using external storage as the main operating system storage, ensure the device's power is off before connecting and removing it.
- Never touch un-insulated terminals or wire unless your power adaptor is disconnected.
- Locate your QEC device as close as possible to the socket outline for easy access and avoid force caused by the entangling of your arms with surrounding cables from the QEC device.
- If your QEC device will not be used for a period of time, make sure it is disconnected from the power source to avoid transient overvoltage damage.

WARNING!

DO NOT ATTEMPT TO OPEN OR TO DISASSEMBLE THE CHASSIS (ENCASING) OF THIS PRODUCT. PLEASE CONTACT YOUR DEALER FOR SERVICING FROM QUALIFIED TECHNICIAN.

CONTENT

Ch. 1

Introduction

1.1 About Ethernet Library

The CPU of the QEC-M series contains a built-in 10/100 Mbps LAN interface, which can access via the Ethernet library. The Arduino Ethernet Shield isn't needed.

This library can serve as either a server accepting incoming connections or a client making outgoing ones. In 86Duino Coding, the library supports up to four concurrent connections (incoming or outgoing or a combination); and in later versions, up to 128 concurrent connections.

1.1.1 What is 86Duino IDE?

The 86Duino integrated development environment (IDE) software makes it easy to write and upload code to 86Duino boards and QEC MDevice. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Arduino IDE, Processing, DJGPP, and other open-source software, which can be downloaded from https://www.qec.tw/software/.

For more information, please visit 86Duino Website.

1.2 Protocol Versions

The library provides IPv4 networking with common client/server features.

Protocol / Feature	Support	Notes
IPv4	Yes	Primary addressing scheme
IPv6	No	IPv4 only
TCP (Server/Client)	Yes	Stream sockets, bidirectional
UDP	Yes	Connectionless datagrams; broadcast supported
DHCP Client	Yes	Auto IP, gateway, DNS
DNS Client	Yes	Connect by hostname
mDNS/Bonjour	Not built-in	Use add-on or custom code
TLS/SSL	Not built-in	Use external solution if needed
WebSocket	Not built-in	Implement at application layer if required

About capacity, the max concurrent sockets and buffer sizes depend on your IDE/runtime build. Document your official limits in an appendix or release notes.

1.3 Communication and Devices

- Topology: Single RJ-45 (10/100 Mbps) to a switch/router.
- Addressing: DHCP by default; fixed IPs recommended for industrial cells.
- Port planning: Common HTTP ports are 80/8080; avoid clashes with other services.
- Performance guidance:
  - Avoid long `delay()` in `loop();` use state machines or time-sliced tasks.
  - Separate packet I/O, parsing, and UI updates to keep each iteration short.
  - For long-lived sessions, check `connected()` regularly and call `stop()` when done.
- Common pitfalls:
  - Adding an Ethernet shield (unnecessary on 86Duino).
  - Treating DHCP/MAC like generic Arduino shields—86Duino typically does not require a user-supplied MAC.
  - Copying examples that rely on blocking calls; adapt them to 86Duino's recommended non-blocking style.

1.4 Functions Table

This section will briefly introduce the Ethernet API usage list.

1.4.1 Ethernet class

The Ethernet class initializes the Ethernet library and network settings.

- `begin()`: Initializes the Ethernet library and network settings.
- `localIP()`: Obtains the IP address of the 86Duino.
- `localMAC()`: Obtains the MAC (Media access control) address of the 86Duino.
- `maintain()`: This function is unused for 86Duino and is reserved here just for the purpose of compatibility with Arduino's Ethernet library.

1.4.2 IPAddress class

The IPAddress class works with local and remote IP addressing.

- `IPAddress()`: Defines an IP address. It can be used to declare both local and remote addresses.

1.4.3 Server class

The Server class creates servers that can send data to and receive data from connected clients (programs running on other computers or devices).

- `EthernetServer()`: Create a server that listens for incoming connections on the specified port.
- `begin()`: Tells the server to begin listening for incoming connections.
- `available()`: Tells the server to begin listening for incoming connections.
- `write()`: Write data to all the clients connected to a server.
- `print()`: Print data to all the clients connected to a server.
- `println()`: Print data, followed by a newline, to all the clients connected to a server.

1.4.4 Client class

The client class creates clients that can connect to servers and send and receive data.

- `EthernetClient()`: Creates a client which can connect to a specified internet IP address and port.
- `if (EthernetClient)`: Indicates if the specified Ethernet client is ready.
- `connected()`: Whether or not the client is connected.
- `connect()`: Connects to a specified IP address and port.
- `write()`: Write data to the server the client is connected to.
- `print()`: Print data to the server that a client is connected to.
- `println()`: Print data, followed by a carriage return and newline, to the server a client is connected to.
- `available()`: Returns the number of bytes available for reading (that is, the amount of data that has been written to the client by the server it is connected to).
- `read()`: Read the next byte received from the server the client is connected to.
- `flush()`: Read the next byte received from the server the client is connected to.
- `stop()`: Disconnect from the server.

1.4.5 EthernetUDP class

The EthernetUDP class enables UDP messages to be sent and received.

- `begin()`: Initializes the ethernet UDP library and network settings.
- `read()`: Reads UDP data from the specified buffer.
- `write()`: Writes UDP data to the remote connection.
- `beginPacket()`: Starts a connection to write UDP data to the remote connection.
- `endPacket()`: Called after writing UDP data to the remote connection.
- `parsePacket()`: Checks for the presence of a UDP packet, and reports the size.
- `available()`: Get the number of bytes (characters) available for reading from the buffer.
- `stop()`: Disconnect from the server. Release any resource being used during the UDP session.
- `remoteIP()`: Gets the IP address of the remote connection.
- `remotePort()`: Gets the port of the remote UDP connection.

Ch. 2

Functions

86Duino Coding IDE 500+.

Ethernet Library.

2.1 Ethernet Class

The Ethernet class initializes the Ethernet library and network settings.

Functions:

- begin()
- localIP()
- localMAC()
- maintain()

begin()

Description

Initializes the Ethernet library and network settings.

This library supports also DHCP. Using `Ethernet.begin(mac)` or `Ethernet.begin()` with the proper network setup, the Ethernet shield will automatically obtain an IP address.

Syntax

`Ethernet.begin();`

`Ethernet.begin(mac);`

`Ethernet.begin(mac, ip);`

`Ethernet.begin(mac, ip, dns);`

`Ethernet.begin(mac, ip, dns, gateway);`

`Ethernet.begin(mac, ip, dns, gateway, subnet);`

Parameters

- `[in] mac`: an array of 6 bytes that contains an MAC (Media access control) address. Since every 86Duino has a fixed MAC address that cannot be changed, this parameter is unused, and is reserved here just for the purpose of compatibility to Arduino's Ethernet library.
- `[in] ip`: the device IP address that you want to set for 86Duino (array of 4 bytes).
- `[in] dns`: the IP address of the DNS server (array of 4 bytes). optional: defaults to the device IP address with the last octet set to 1.
- `[in] gateway`: the IP address of the network gateway (array of 4 bytes). optional: defaults to the device IP address with the last octet set to 1.
- `[in] subnet`: the subnet mask of the network (array of 4 bytes). optional: defaults to 255.255.255.0.

Returns

The DHCP version of this function, `Ethernet.begin(mac)` or `Ethernet.begin()`, returns an int: 1 on a successful DHCP connection, 0 on failure.

The other versions don't return anything.

Example

```
#include <Ethernet.h>

byte mac[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }; // MAC
byte ip[] = { 10, 0, 0, 177 }; //the IP address

void setup() {
  // put your setup code here, to run once:
  Ethernet.begin(mac, ip);
}

void loop () {
  // put your main code here, to run repeatedly:
```

```
}
```

localIP()

Description

Obtains the IP address of the 86Duino.

Useful when the address is auto-assigned through DHCP.

Syntax

`Ethernet.localIP();`

Parameters

None.

Returns

The IP address.

Example

```
#include <Ethernet.h>

void setup() {
  // start the serial library:
  Serial.begin(9600);
  // start the Ethernet connection:
  if (Ethernet.begin() == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // no point in carrying on, so do nothing forevermore:
    for (;;)
      ;
  }
  // print your local IP address:
  Serial.println(Ethernet.localIP());
}

void loop() {

}
```

localMAC()

Description

Obtains the MAC (Media access control) address of the 86Duino.

This function is available from 86Duino Coding 102.

Syntax

`Ethernet.localMAC();`

Parameters

None.

Returns

An array of 6 bytes that contains the MAC address of the 86Duino.

Example

```
#include <Ethernet.h>

void setup() {
  // start the serial library:
  Serial.begin(9600);
  // start the Ethernet connection:
  if (Ethernet.begin() == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // no point in carrying on, so do nothing forevermore:
    for (;;)
      ;
  }
  // print your MAC address:
  byte *mac = Ethernet.localMAC();
  for (int i = 0; i < 6; i++)
  {
    Serial.print(mac[i], HEX);
    Serial.print(" ");
  }
}

void loop() {

}
```

maintain()

Description

This function is unused for 86Duino and is reserved here just for the purpose of compatibility with Arduino's Ethernet library.

Syntax

```
Ethernet.maintain()
```

Parameters

None.

Returns

Byte (always 0).

2.2 IPAddress Class

The IPAddress class works with local and remote IP addressing.

Functions:

- IPAddress()

IPAddress()

Description

Defines an IP address. It can be used to declare both local and remote addresses.

Syntax

`IPAddress(address);`

Parameters

- `[in] address`: a comma-delimited list representing the address (4 bytes, ex. 192, 168, 1, 1).

Returns

None.

Example

```
#include <Ethernet.h>

// network configuration. dns server, gateway and subnet are optional.
byte mac[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };

// the dns server ip
IPAddress dnServer(192, 168, 0, 1);
// the router's gateway address:
IPAddress gateway(192, 168, 0, 1);
// the subnet:
IPAddress subnet(255, 255, 255, 0);

// the IP address is dependent on your network
IPAddress ip(192, 168, 0, 2);

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

  // initialize the ethernet device
  Ethernet.begin(mac, ip, dnServer, gateway, subnet);
  // print out the IP address
  Serial.print("IP = ");
  Serial.println(Ethernet.localIP());
}

void loop() {
 // put your main code here, to run repeatedly:
}
```

2.3 Server Class

The Server class creates servers that can send data to and receive data from connected clients (programs running on other computers or devices).

Functions:

- `EthernetServer()`
- `begin()`
- `available()`
- `write()`
- `print()`
- `println()`

EthernetServer()

Description

Create a server that listens for incoming connections on the specified port.

Syntax

`EthernetServer(port);`

Parameters

- `[in] port`: the port to listen on (int).

Returns

None.

Example

```
#include <Ethernet.h>

// network configuration.  gateway and subnet are optional.
// the IP address:
byte ip[] = { 10, 0, 0, 177 };
// the router's gateway address:
byte gateway[] = { 10, 0, 0, 1 };
// the subnet:
byte subnet[] = { 255, 255, 0, 0 };

// telnet defaults to port 23
EthernetServer server = EthernetServer(23);

void setup() {
  // initialize the ethernet device
  Ethernet.begin(NULL, ip, gateway, subnet);
  // start listening for clients
  server.begin();
}

void loop() {
  // if an incoming client connects, there will be bytes available to read:
  EthernetClient client = server.available();
  if (client == true) {
    // read bytes from the incoming client and write them back
    // to any clients connected to the server:
    server.write(client.read());
  }
}
```

begin()

Description

Tells the server to begin listening for incoming connections.

Syntax

`EthernetServer server = EthernetServer(int);`

`server.begin();`

Parameters

None.

Returns

None.

Example

```
#include <Ethernet.h>
// network configuration.  gateway and subnet are optional.
// the IP address:
byte ip[] = { 10, 0, 0, 177 };
// the router's gateway address:
byte gateway[] = { 10, 0, 0, 1 };
// the subnet:
byte subnet[] = { 255, 255, 0, 0 };

// telnet defaults to port 23
EthernetServer server = EthernetServer(23);

void setup() {
  // initialize the ethernet device
  Ethernet.begin(NULL, ip, gateway, subnet);
  // start listening for clients
  server.begin();
}

void loop() {
  // if an incoming client connects, there will be bytes available to read:
  EthernetClient client = server.available();
  if (client == true) {
    // read bytes from the incoming client and write them back
    // to any clients connected to the server:
    server.write(client.read());
  }
}
```

available()

Description

Tells the server to begin listening for incoming connections.

Gets a client that is connected to the server and has data available for reading. The connection persists when the returned client object goes out of scope; you can close it by calling `client.stop()`.

`available()` inherits from the Stream utility class.

Syntax

```
EthernetServer server = EthernetServer(int);
server.available();
```

Parameters

None.

Returns

A Client object; if no Client has data available for reading, this object will evaluate to false in an if-statement (see the example below).

Example

```
#include <Ethernet.h>

//the IP address:
byte ip[] = { 10, 0, 0, 177 };
// the router's gateway address:
byte gateway[] = { 10, 0, 0, 1 };
// the subnet:
byte subnet[] = { 255, 255, 0, 0 };

// telnet defaults to port 23
EthernetServer server = EthernetServer(23);

void setup()
{
  // initialize the ethernet device
  Ethernet.begin(NULL, ip, gateway, subnet);

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

void loop()
{
  // if an incoming client connects, there will be bytes available to read:
```

```
  EthernetClient client = server.available();
  if (client == true) {
    // read bytes from the incoming client and write them back
    // to any clients connected to the server:
    server.write(client.read());
  }
}
```

write()

Description

Write data to all the clients connected to a server. This data is sent as a byte or series of bytes.

Syntax

```
EthernetServer server = EthernetServer(int);
server.write(val);
server.write(buf, len);
```

Parameters

- `[in] val`: a value to send as a single byte (byte or char).
- `[in] buf`: an array to send as a series of bytes (byte or char).
- `[in] len`: the length of the buffer.

Returns

byte (`write()` returns the number of bytes written. It is not necessary to read this.)

Example

```
#include <Ethernet.h>

// network configuration.  gateway and subnet are optional.

byte mac[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
//the IP address:
byte ip[] = { 10, 0, 0, 177 };
// the router's gateway address:
byte gateway[] = { 10, 0, 0, 1 };
// the subnet:
byte subnet[] = { 255, 255, 0, 0 };

// telnet defaults to port 23
EthernetServer server = EthernetServer(23);

void setup()
{
  // initialize the ethernet device
  Ethernet.begin(mac, ip, gateway, subnet);

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

void loop()
{
```

```
  // if an incoming client connects, there will be bytes available to read:
  EthernetClient client = server.available();
  if (client == true) {
    // read bytes from the incoming client and write them back
    // to any clients connected to the server:
    server.write(client.read());
  }
}
```

print()

Description

Print data to all the clients connected to a server. Prints numbers as a sequence of digits, each an ASCII character (e.g. the number 123 is sent as the three characters ‘1’, ‘2’, ‘3’).

Syntax

```
EthernetServer server = EthernetServer(int);
server.print(data);
server.print(data, BASE);
```

Parameters

- `[in] data`: the data to print (char, byte, int, long, or string).
- `[in] BASE (optional)`: the base in which to print numbers: BIN for binary (base 2), DEC for decimal (base 10), OCT for octal (base 8), HEX for hexadecimal (base 16).

Returns

byte (`print()` will return the number of bytes written, though reading that number is optional)

println()

Description

Print data, followed by a newline, to all the clients connected to a server. Prints numbers as a sequence of digits, each an ASCII character (e.g. the number 123 is sent as the three characters ‘1’, ‘2’, ‘3’).

Syntax

```
EthernetServer server = EthernetServer(int);
server.println()
server.println(data)
server.println(data, BASE)
```

Parameters

- `[in] data (optional)`: the data to print (char, byte, int, long, or string).
- `[in] BASE (optional)`: the base in which to print numbers: BIN for binary (base 2), DEC for decimal (base 10), OCT for octal (base 8), HEX for hexadecimal (base 16).

Returns

byte (`println()` will return the number of bytes written, though reading that number is optional)

2.4 Client Class

The client class creates clients that can connect to servers and send and receive data.

Functions:

- `EthernetClient()`
- `if (EthernetClient)`
- `connected()`
- `connect()`
- `write()`
- `print()`
- `println()`
- `available()`
- `read()`
- `flush()`
- `stop()`

EthernetClient()

Description

Creates a client which can connect to a specified internet IP address and port (defined in the `client.connect()` function).

Syntax

`EthernetClient();`

Parameters

None.

Returns

None.

Example

```
#include <Ethernet.h>

byte ip[] = { 10, 0, 0, 177 };
byte server[] = { 64, 233, 187, 99 }; // Google

EthernetClient client;

void setup()
{
  Ethernet.begin(NULL, ip);
  Serial.begin(9600);

  delay(1000);

  Serial.println("connecting...");

  if (client.connect(server, 80)) {
    Serial.println("connected");
    client.println("GET /search?q=arduino HTTP/1.0");
    client.println();
  } else {
    Serial.println("connection failed");
  }
}

void loop()
{
  if (client.available()) {
    char c = client.read();
```

```
    Serial.print(c);
  }

  if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();
    for (;;)
      ;
  }
}
```

if (EthernetClient)

Description

Indicates if the specified Ethernet client is ready.

Syntax

`EthernetClient client;`

`if (client);`

Parameters

None.

Returns

Boolean: returns true if the specified client is available.

Example

```
#include <Ethernet.h>

byte ip[] = { 10, 0, 0, 177 };
byte server[] = { 64, 233, 187, 99 }; // Google

EthernetClient client;

void setup()
{
  Ethernet.begin(NULL, ip);
  Serial.begin(9600);

  delay(1000);

  Serial.println("connecting...");

  if (client.connect(server, 80)) {
    Serial.println("connected");
    client.println("GET /search?q=arduino HTTP/1.0");
    client.println();
  } else {
    Serial.println("connection failed");
  }
}

void loop()
{
  if (client.available()) {
    char c = client.read();
```

```
    Serial.print(c);
  }

  if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();
    for (;;)
      ;
  }
}
```

connected()

Description

Whether or not the client is connected. Note that a client is considered connected if the connection has been closed but there is still unread data.

Syntax

`EthernetClient client;`
`client.connected();`

Parameters

None.

Returns

Returns true if the client is connected, false if not.

Example

```
#include <Ethernet.h>

byte mac[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
byte ip[] = { 10, 0, 0, 177 };
byte server[] = { 64, 233, 187, 99 }; // Google

EthernetClient client;

void setup()
{
  Ethernet.begin(mac, ip);
  Serial.begin(9600);
  client.connect(server, 80);
  delay(1000);

  Serial.println("connecting...");

  if (client.connected()) {
    Serial.println("connected");
    client.println("GET /search?q=arduino HTTP/1.0");
    client.println();
  } else {
    Serial.println("connection failed");
  }
}

void loop()
{
```

```
  if (client.available()) {
    char c = client.read();
    Serial.print(c);
  }

  if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();
    for(;;)
      ;
  }
}
```

connect()

Description

Connects to a specified IP address and port. The return value indicates success or failure. Also supports DNS lookups when using a domain name.

Syntax

`EthernetClient client;`
`client.connect();`
`client.connect(ip, port);`
`client.connect(URL, port);`

Parameters

- `[in] ip`: the IP address that the client will connect to (array of 4 bytes).
- `[in] URL`: the domain name the client will connect to (string, ex.:"arduino.cc").
- `[in] port`: the port that the client will connect to (int).

Returns

Returns true if the connection succeeds, false if not.

Example

```
#include <Ethernet.h>

byte ip[] = { 10, 0, 0, 177 };
byte server[] = { 64, 233, 187, 99 }; // Google

EthernetClient client;

void setup()
{
  Ethernet.begin(NULL, ip);
  Serial.begin(9600);

  delay(1000);

  Serial.println("connecting...");

  if (client.connect(server, 80)) {
    Serial.println("connected");
    client.println("GET /search?q=arduino HTTP/1.0");
    client.println();
  } else {
    Serial.println("connection failed");
  }
}
```

```
void loop()
{
  if (client.available()) {
    char c = client.read();
    Serial.print(c);
  }

  if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();
    for(;;)
      ;
  }
}
```

write()

Description

Write data to the server the client is connected to. This data is sent as a byte or series of bytes.

Syntax

`EthernetClient client;`

`client.write(val);`

`client.write(buf, len);`

Parameters

- `[in] val`: a value to send as a single byte (byte or char).
- `[in] buf`: an array to send as a series of bytes (byte or char).
- `[in] len`: the length of the buffer.

Returns

byte (`write()` returns the number of bytes written. It is not necessary to read this value.)

Example

```
#include <Ethernet.h>

byte ip[] = { 10, 0, 0, 177 };
byte server[] = { 64, 233, 187, 99 }; // Google

EthernetClient client;

void setup()
{
  Ethernet.begin(NULL, ip);
  Serial.begin(9600);

  delay(1000);

  Serial.println("connecting...");

  if (client.connect(server, 80)) {
    Serial.println("connected");
    client.println("GET /search?q=arduino HTTP/1.0");
    client.println();
  } else {
    Serial.println("connection failed");
  }
}

void loop()
```

```
{
  if (client.available()) {
    char c = client.read();
    Serial.print(c);
  }

  if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();
    for(;;)
      ;
  }
}
```

print()

Description

Print data to the server that a client is connected to. Prints numbers as a sequence of digits, each an ASCII character (e.g. the number 123 is sent as the three characters ‘1’, ‘2’, ‘3’).

Syntax

```
EthernetClient client;
client.print(data);
client.print(data, BASE);
```

Parameters

- `[in] data`: the data to print (char, byte, int, long, or string)
- `[in] BASE (optional)`: the base in which to print numbers: DEC for decimal (base 10), OCT for octal (base 8), HEX for hexadecimal (base 16).

Returns

Byte: returns the number of bytes written, though reading that number is optional.

println()

Description

Print data, followed by a carriage return and newline, to the server a client is connected to. Prints numbers as a sequence of digits, each an ASCII character (e.g. the number 123 is sent as the three characters ‘1’, ‘2’, ‘3’).

Syntax

```
EthernetClient client;
client.println();
client.println(data);
client.print(data, BASE);
```

Parameters

- `[in] data (optional)`: the data to print (char, byte, int, long, or string)
- `[in] BASE (optional)`: the base in which to print numbers: DEC for decimal (base 10), OCT for octal (base 8), HEX for hexadecimal (base 16).

Returns

Byte: return the number of bytes written, though reading that number is optional.

available()

Description

Returns the number of bytes available for reading (that is, the amount of data that has been written to the client by the server it is connected to).

`available()` inherits from the Stream utility class.

Syntax

```
EthernetClient client;
client.available();
```

Parameters

None.

Returns

The number of bytes available.

Example

```
#include <Ethernet.h>

byte ip[] = { 10, 0, 0, 177 };
byte server[] = { 64, 233, 187, 99 }; // Google

EthernetClient client;

void setup()
{
  Ethernet.begin(NULL, ip);
  Serial.begin(9600);

  delay(1000);

  Serial.println("connecting...");

  if (client.connect(server, 80)) {
    Serial.println("connected");
    client.println("GET /search?q=arduino HTTP/1.0");
    client.println();
  } else {
    Serial.println("connection failed");
  }
}

void loop()
{
```

```
  if (client.available()) {
    char c = client.read();
    Serial.print(c);
  }

  if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting.");
    client.stop();
    for (;;)
      ;
  }
}
```

read()

Description

Read the next byte received from the server the client is connected to (after the last call to `read()`).

`read()` inherits from the Stream utility class.

Syntax

```
EthernetClient client;
client.read();
```

Parameters

None.

Returns

The next byte (or character), or -1 if none is available.

flush()

Description

Discard any bytes that have been written to the client but not yet read.

`flush()` inherits from the Stream utility class.

Syntax

```
EthernetClient client;
client.flush();
```

Parameters

None.

Returns

None.

stop()

Description

Disconnect from the server.

Syntax

```
EthernetClient client;
client.stop();
```

Parameters

None.

Returns

None.

2.5 EthernetUDP class

The EthernetUDP class enables UDP messages to be sent and received.

Functions:

- begin()
- read()
- write()
- beginPacket()
- endPacket()
- parsePacket()
- available()
- stop()
- remoteIP()
- remotePort()

begin()

Description

Initializes the ethernet UDP library and network settings.

Syntax

`EthernetUDP Udp;`

`Udp.begin(localPort);`

Parameters

- `[in] localPort`: the local port to listen on (int).

Returns

None.

Example

```
#include <Ethernet.h>
#include <EthernetUdp.h>

// Enter an IP address for your controller below.
// The IP address will be dependent on your local network:
IPAddress ip(192, 168, 1, 177);

unsigned int localPort = 8888;      // local port to listen on
// An EthernetUDP instance to let us send and receive packets over UDP
EthernetUDP Udp;

void setup() {
  // start the Ethernet and UDP:
  Ethernet.begin(NULL, ip);
  Udp.begin(localPort);
}

void loop() {

}
```

read()

Description

Reads UDP data from the specified buffer. If no arguments are given, it will return the next character in the buffer.

This function can only be successfully called after `Udp.parsePacket()`.

Syntax

`EthernetUDP Udp;`

`Udp.read();`

`Udp.read(packetBuffer, MaxSize);`

Parameters

- `[in] packetBuffer`: buffer to hold incoming packets (char).
- `[in] MaxSize`: maximum size of the buffer (int).

Returns

Char: returns the characters in the buffer.

Example

```
#include <Ethernet.h>
#include <EthernetUdp.h>

// Enter an IP address for your controller below.
// The IP address will be dependent on your local network:
IPAddress ip(192, 168, 1, 177);

unsigned int localPort = 8888;      // local port to listen on

// An EthernetUDP instance to let us send and receive packets over UDP
EthernetUDP Udp;

char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; //buffer to hold incoming packet,

void setup() {
  // start the Ethernet and UDP:
  Ethernet.begin(NULL, ip);
  Udp.begin(localPort);
}

void loop() {
  int packetSize = Udp.parsePacket();
  if (packetSize)
  {
    Serial.print("Received packet of size ");
```

```
    Serial.println(packetSize);
    Serial.print("From ");
    IPAddress remote = Udp.remoteIP();
    for (int i = 0; i < 4; i++)
    {
      Serial.print(remote[i], DEC);
      if (i < 3)
      {
        Serial.print(".");
      }
    }
    Serial.print(", port ");
    Serial.println(Udp.remotePort());

    // read the packet into packetBufffer
    Udp.read(packetBuffer, UDP_TX_PACKET_MAX_SIZE);
    Serial.println("Contents:");
    Serial.println(packetBuffer);
  }
}
```

write()

Description

Writes UDP data to the remote connection. Must be wrapped between `beginPacket()` and `endPacket()`.

`beginPacket()` initializes the packet of data, it is not sent until `endPacket()` is called. (Remarks: the maximum size of the packet is 512-byte in 86Duino, in contrast with 24-byte in Arduino.)

Syntax

`EthernetUDP Udp;`

`Udp.write(message);`

`Udp.write(buffer, size);`

Parameters

- `[in] message`: the outgoing message (char).
- `[in] buffer`: an array to send as a series of bytes (byte or char).
- `[in] size`: the length of the buffer.

Returns

Byte: returns the number of characters sent. This does not have to be read.

Example

```
#include <Ethernet.h>
#include <EthernetUdp.h>
// Enter an IP address for your controller below.
// The IP address will be dependent on your local network:
IPAddress ip(192, 168, 1, 177);
unsigned int localPort = 8888;      // local port to listen on
// An EthernetUDP instance to let us send and receive packets over UDP
EthernetUDP Udp;

void setup() {
  // start the Ethernet and UDP:
  Ethernet.begin(NULL, ip);
  Udp.begin(localPort);
}

void loop() {
  Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
  Udp.write("hello");
  Udp.endPacket();
}
```

beginPacket()

Description

Starts a connection to write UDP data to the remote connection.

Syntax

`EthernetUDP Udp;`

`Udp.beginPacket(remoteIP, remotePort);`

Parameters

- `[in] remoteIP`: the IP address of the remote connection (4 bytes).
- `[in] remotePort`: the port of the remote connection (int).

Returns

None.

Example

```
#include <Ethernet.h>
#include <EthernetUdp.h>

// Enter an IP address for your controller below.
// The IP address will be dependent on your local network:
IPAddress ip(192, 168, 1, 177);

unsigned int localPort = 8888;      // local port to listen on

// An EthernetUDP instance to let us send and receive packets over UDP
EthernetUDP Udp;

void setup() {
  // start the Ethernet and UDP:
  Ethernet.begin(NULL,ip);
  Udp.begin(localPort);
}

void loop() {
  Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
  Udp.write("hello");
  Udp.endPacket();
}
```

endPacket()

Description

Called after writing UDP data to the remote connection.

Syntax

`EthernetUDP Udp;`

`Udp.endPacket();`

Parameters

None.

Returns

None.

Example

```
#include <Ethernet.h>
#include <EthernetUdp.h>

// Enter an IP address for your controller below.
// The IP address will be dependent on your local network:
IPAddress ip(192, 168, 1, 177);

unsigned int localPort = 8888;      // local port to listen on

// An EthernetUDP instance to let us send and receive packets over UDP
EthernetUDP Udp;

void setup() {
  // start the Ethernet and UDP:
  Ethernet.begin(NULL,ip);
  Udp.begin(localPort);
}

void loop() {
  Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
  Udp.write("hello");
  Udp.endPacket();
}
```

parsePacket()

Description

Checks for the presence of a UDP packet, and reports the size.

`parsePacket()` must be called before reading the buffer with `Udp.read()`.

Syntax

```
EthernetUDP Udp;
Udp.parsePacket();
```

Parameters

None.

Returns

Int: the size of a received UDP packet.

Example

```
#include <Ethernet.h>
#include <EthernetUdp.h>

// Enter an IP address for your controller below.
// The IP address will be dependent on your local network:
IPAddress ip(192, 168, 1, 177);

unsigned int localPort = 8888;      // local port to listen on

// An EthernetUDP instance to let us send and receive packets over UDP
EthernetUDP Udp;

void setup() {
  // start the Ethernet and UDP:
  Ethernet.begin(NULL, ip);
  Udp.begin(localPort);

  Serial.begin(9600);
}

void loop() {
  // if there's data available, read a packet
  int packetSize = Udp.parsePacket();
  if (packetSize)
  {
    Serial.print("Received packet of size ");
    Serial.println(packetSize);
  }
```

```
  delay(10);
}
```

available()

Description

Get the number of bytes (characters) available for reading from the buffer. This is data that's already arrived. This function can only be successfully called after `Udp.parsePacket()`. `available()` inherits from the Stream utility class.

Syntax

`EthernetUDP Udp;`

`Udp.available();`

Parameters

None.

Returns

The number of bytes available to read.

Example

```
#include <Ethernet.h>
#include <EthernetUdp.h>

// Enter an IP address for your controller below.
// The IP address will be dependent on your local network:
IPAddress ip(192, 168, 1, 177);

unsigned int localPort = 8888;      // local port to listen on

// An EthernetUDP instance to let us send and receive packets over UDP
EthernetUDP Udp;

char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; //buffer to hold incoming packet,

void setup() {
  // start the Ethernet and UDP:
  Ethernet.begin(NULL, ip);
  Udp.begin(localPort);
}

void loop() {
  int packetSize = Udp.parsePacket();

  if (Udp.available())
  {
    Serial.print("Received packet of size ");
    Serial.println(packetSize);
```

```
    Serial.print("From ");
    IPAddress remote = Udp.remoteIP();
    for (int i = 0; i < 4; i++)
    {
      Serial.print(remote[i], DEC);
      if (i < 3)
      {
        Serial.print(".");
      }
    }
    Serial.print(", port ");
    Serial.println(Udp.remotePort());

    // read the packet into packetBufffer
    Udp.read(packetBuffer, UDP_TX_PACKET_MAX_SIZE);
    Serial.println("Contents:");
    Serial.println(packetBuffer);
  }
}
```

stop()

Description

Disconnect from the server. Release any resource being used during the UDP session.

Syntax

```
EthernetUDP Udp;
Udp.stop();
```

Parameters

None.

Returns

None.

remoteIP()

Description

Gets the IP address of the remote connection.

This function must be called after `Udp.parsePacket()`.

Syntax

`EthernetUDP Udp;`

`Udp.remoteIP();`

Parameters

None.

Returns

4 bytes: the IP address of the remote connection.

Example

```
#include <Ethernet.h>
#include <EthernetUdp.h>

// Enter an IP address for your controller below.
// The IP address will be dependent on your local network:
IPAddress ip(192, 168, 1, 177);

unsigned int localPort = 8888;      // local port to listen on

// An EthernetUDP instance to let us send and receive packets over UDP
EthernetUDP Udp;

void setup() {
  // start the Ethernet and UDP:
  Ethernet.begin(NULL, ip);
  Udp.begin(localPort);
}

void loop() {
  int packetSize = Udp.parsePacket();

  if (packetSize)
  {
    Serial.print("Received packet of size ");
    Serial.println(packetSize);
    Serial.print("From IP : ");

    IPAddress remote = Udp.remoteIP();
```

```
    //print out the remote connection's IP address
    Serial.print(remote);

    Serial.print(" on port : ");
    //print out the remote connection's port
    Serial.println(Udp.remotePort());
  }
}
```

remotePort()

Description

Gets the port of the remote UDP connection.

This function must be called after `UDP.parsePacket()`.

Syntax

```
EthernetUDP Udp;
Udp.remotePort();
```

Parameters

None.

Returns

Int: the port of the UDP connection to a remote host.

Example

```
#include <Ethernet.h>
#include <EthernetUdp.h>

// Enter an IP address for your controller below.
// The IP address will be dependent on your local network:
IPAddress ip(192, 168, 1, 177);

unsigned int localPort = 8888;      // local port to listen on

// An EthernetUDP instance to let us send and receive packets over UDP
EthernetUDP Udp;

char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; //buffer to hold incoming packet,

void setup() {
  // start the Ethernet and UDP:
  Ethernet.begin(NULL, ip);
  Udp.begin(localPort);
}

void loop() {
  int packetSize = Udp.parsePacket();

  if (packetSize)
  {
    Serial.print("Received packet of size ");
    Serial.println(packetSize);
    Serial.print("From ");
```

```
    IPAddress remote = Udp.remoteIP();
    for (int i = 0; i < 4; i++)
    {
      Serial.print(remote[i], DEC);
      if (i < 3)
      {
        Serial.print(".");
      }
    }
    Serial.print(", port ");
    Serial.println(Udp.remotePort());

    // read the packet into packetBufffer
    Udp.read(packetBuffer, UDP_TX_PACKET_MAX_SIZE);
    Serial.println("Contents:");
    Serial.println(packetBuffer);
  }
}
```

Ch. 3

Example

3.1 Ethernet Example

The following are examples of the Ethernet library from the Arduino Tutorial that can work on the 86Duino boards (note that on 86Duino, you don't need the Arduino Ethernet Shield shown in these examples):

- ChatServer: set up a simple chat server.
- WebClient: make an HTTP request.
- WebClientRepeating: Make repeated HTTP requests.
- WebServer: host a simple HTML page that displays analog sensor values.
- BarometricPressureWebServer: outputs the values from a barometric pressure sensor as a web page.
- UDPSendReceiveString: Send and receive text strings via UDP.
- UdpNtpClient: Query a Network Time Protocol (NTP) server using UDP.
- DnsWebClient: DNS and DHCP-based Web client.
- DhcpChatServer: A simple DHCP Chat Server.
- DhcpAddressPrinter: Get an IP address via DHCP and print it out.
- TelnetClient: A simple Telnet client.

3.2 Hack

We employ the SwsSock library of Software Systems to implement the Ethernet library of 86Duino. If interested in directly using SwsSock in your 86Duino sketches, you may refer to the Web page of SwsSock for related information.

Warranty

This product is warranted to be in good working order for a period of one year from the date of purchase. Should this product fail to be in good working order at any time during this period, we will, at our option, replace or repair it at no additional charge except as set forth in the following terms. This warranty does not apply to products damaged by misuse, modifications, accident or disaster. Vendor assumes no liability for any damages, lost profits, lost savings or any other incidental or consequential damage resulting from the use, misuse of, originality to use this product. Vendor will not be liable for any claim made by any other related party. Return authorization must be obtained from the vendor before returned merchandise will be accepted. Authorization can be obtained by calling or faxing the vendor and requesting a Return Merchandise Authorization (RMA) number. Returned goods should always be accompanied by a clear problem description.