

```

// =====
// this is the pfodParser.h file with the class renamed pfodParser_codeGenerated and with comments, constants and un-
used methods removed
class pfodParser_codeGenerated: public Print {
public:
    pfodParser_codeGenerated();
    void connect(Stream* ioPtr); void closeConnection(); byte parse(); byte* getCmd(); byte* getFirstArg();
    byte getArgsCount(); byte* parseLong(byte* idxPtr, long *result); size_t write(uint8_t c); void flush();
    void init(); byte parse(byte in); Stream* getPfodAppStream();
private:
    Stream* io; byte argsCount; byte argsIdx; byte parserState; byte args[255];
};
//===== end of pfodParser_codeGenerated.h

#include <LWiFi.h>
#include <LWiFiServer.h>
#include <LWiFiClient.h>

#define WIFI_AP "FCC-WfiTest"
#define WIFI_PASSWORD ""
#define WIFI_AUTH LWIFI_OPEN // choose from LWIFI_OPEN, LWIFI_WPA, or LWIFI_WEP according to your
WiFi AP configuration

LWiFiServer server(4989);
LWiFiClient client;
pfodParser_codeGenerated parser; // create a parser to handle the pfod messages
boolean alreadyConnected = false; // whether or not the client was connected previously

// give the board pins names, if you change the pin number here you will change the pin controlled
int cmd_A_pin = 3; // name the output pin for 'Pin D3 is '

void setup(){
    LWiFi.begin();
    Serial.begin(9600);
    for (int i = 10; i > 0; i--) {
        delay(1000);
        Serial.print(i);
        Serial.print(' ');
    }
    Serial.println();
    // keep retrying until connected to AP
    Serial.println("Connecting to AP");
    while (0 == LWiFi.connect(WIFI_AP, LWiFiLoginInfo(WIFI_AUTH, WIFI_PASSWORD)))
    {
        Serial.println("connect returned 0 try again in 1sec");
        delay(1000);
    }

    printWifiStatus();
    //pinMode(cmd_A_pin, INPUT_PULLUP);
    pinMode(cmd_A_pin, OUTPUT); // output for 'Pin D3 is ' is initially LOW, uncomment line above if you want it
initially HIGH

    Serial.println("Start Server");
    server.begin();
    Serial.println("Server Started");
    client = server.available(); // evaluates to false if no connection
}

void loop() {

```



```

void printWifiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(LWiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = LWiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  Serial.print("subnet mask: ");
  Serial.println(LWiFi.subnetMask());

  Serial.print("gateway IP: ");
  Serial.println(LWiFi.gatewayIP());

  // print the received signal strength:
  long rssi = LWiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}

void sendMainMenu() {
  parser.print(F("{:}")); // start a Menu screen pfod message
  send_menuContents(); // send the menu contents
  parser.print(F("}")); // close pfod message
}

void sendMainMenuUpdate() {
  parser.print(F(":{:}")); // start an Update Menu pfod message
  send_menuContents(); // send the menu contents
  parser.print(F("}")); // close pfod message
}

// modify this method if you need to update the menu to reflect state changes
void send_menuContents() {
  // send menu prompt
  parser.print(F("<w><+5><b>Using Ethernet Shield to turn D3 on and off"));
  // send menu items
  parser.print(F("|A~<+4><b>Pin D3 is\n`"));
  parser.print(digitalRead(cmd_A_pin)); // read current output state 0 Low or 1 High
  parser.print(F("~~Low\\High")); // Note the \\ inside the ""s to send \
  // ===== end of menu item =====
}

//=====
/* You can remove from here on if you have the pfodParser library installed
 * and add
#include <pfodParser.h>
 * at the top of this file
 * and replace the line
pfodParser_codeGenerated parser; // create a parser to handle the pfod messages
 * with
pfodParser parser;
 */
// this is the pfodParser.cpp file with the class renamed pfodParser_codeGenerated and with comments, constants and
un-used methods removed
pfodParser_codeGenerated::pfodParser_codeGenerated() {
  io = NULL; init();
}
void pfodParser_codeGenerated::init() {

```

```

    argsCount = 0; argsIdx = 0; args[0] = 0; args[1] = 0; parserState = ((byte)0xff);
}
void pfodParser_codeGenerated::connect(Stream* ioPtr) {
    init(); io = ioPtr;
}
void pfodParser_codeGenerated::closeConnection() {
    init();
}
Stream* pfodParser_codeGenerated::getPfodAppStream() {
    return io;
}
size_t pfodParser_codeGenerated::write(uint8_t c) {
    if (!io) {
        return 1; // cannot write if io null but just pretend to
    }
    return io->write(c);
}
void pfodParser_codeGenerated::flush() {
    if (!io) {
        return ; // cannot write if io null but just pretend to
    }
    io->flush();
}
byte* pfodParser_codeGenerated::getCmd() {
    return args;
}
byte* pfodParser_codeGenerated::getFirstArg() {
    byte* idxPtr = args;
    while ( *idxPtr != 0) {
        ++idxPtr;
    }
    if (argsCount > 0) {
        ++idxPtr;
    }
    return idxPtr;
}
byte pfodParser_codeGenerated::getArgsCount() {
    return argsCount;
}
byte pfodParser_codeGenerated::parse() {
    byte rtn = 0;
    if (!io) {
        return rtn;
    }
    while (io->available()) {
        int in = io->read();
        rtn = parse((byte)in);
        if (rtn != 0) {
            // found msg
            if (rtn == '!') {
                closeConnection();
            }
        }
        return rtn;
    }
}
return rtn;
}
byte pfodParser_codeGenerated::parse(byte in) {
    if ((parserState == ((byte)0xff)) || (parserState == ((byte)'}'))) {
        parserState = ((byte)0xff);
        if (in == ((byte){'})) {
            init();
            parserState = ((byte){'});
        }
    }
}

```

```

    }
    return 0;
}
if ((argsIdx >= (255 - 2)) &&
    (in != ((byte)''))) {
    init();
    return 0;
}
if (parserState == ((byte)')) {
    parserState = ((byte)0);
}
if ((in == ((byte)')) || (in == ((byte)')) || (in == ((byte)~)) || (in == ((byte)''))) {
    args[argsIdx++] = 0;
    if (parserState == ((byte)0xfe)) {
        argsCount++;
    }
    if (in == ((byte)')) {
        parserState = ((byte)'); // reset state
        return args[0];
    } else {
        parserState = ((byte)0xfe);
    }
    return 0;
}
args[argsIdx++] = in;
return 0;
}
byte* pfodParser_codeGenerated::parseLong(byte* idxPtr, long *result) {
    long rtn = 0;
    boolean neg = false;
    while ( *idxPtr != 0) {
        if (*idxPtr == '-') {
            neg = true;
        } else {
            rtn = (rtn << 3) + (rtn << 1);
            rtn = rtn + (*idxPtr - '0');
        }
        ++idxPtr;
    }
    if (neg) {
        rtn = -rtn;
    }
    *result = rtn;
    return ++idxPtr;
}
// ===== end generated code =====

```