```
#include <LBT.h>
#include <LBTServer.h>
// 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

pfodParser_codeGenerated parser; // create a parser to handle the pfod messages

// give the board pins names, if you change the pin number here you will change the pin controlled
int cmd_A_pin = 13; // name the output pin for 'Led is '
//bool LBTServerRunning = false;
boolean alreadyConnected = false; // whether or not the client was connected previously


void setup()
{
  Serial.begin(9600);
  for (int i = 10; i > 0; i--) {
    delay(1000);
//    Serial.print(i);
//    Serial.print(' ');
  }
//  Serial.println();
  startServer();
  //pinMode(cmd_A_pin, INPUT_PULLUP);
  pinMode(cmd_A_pin, OUTPUT); // output for 'Led is ' is initially LOW, uncomment line above if you want it initially
HIGH


}

void startServer() {
//  Serial.println("Bluetooth Server starting");
  bool LBTServerRunning = LBTServer.begin((uint8_t*)"pfodLinkItONE");
  if ( !LBTServerRunning )  {
    // don't do anything else
    while (true) {
      Serial.println(F("Cannot begin Bluetooth Server successfully"));
      delay(5000);
    }
  }  else  {
//    Serial.println("Bluetooth Server begin successfully");
  }
}

void loop() {
  if (!LBTServer.connected()) {
    if (alreadyConnected) {
      // client closed so clean up
      Serial.println("BT not connected but was connected");
      closeConnection(parser.getPfodAppStream());
    }
    alreadyConnected = false;
  //  Serial.println("BT not connected wait for connection");
```

```
    // waiting for Spp Client to connect
   // bool connected = LBTServer.accept(1); // fails to connect after <1hr
   // bool connected = LBTServer.accept(5); // fails to connect after <1hr
    bool connected = LBTServer.accept(20); // also fails to connect after <1hr
    if (!connected) {
    //  Serial.println("No connection request yet");
      // don't do anything else
    } else {
    //  Serial.println("Connected");
    }
  }

  if (LBTServer.connected() && (!alreadyConnected)) {
   // Serial.println("BT connected set up parser");
    parser.connect(&LBTServer); // sets new io stream to read from and write to
    alreadyConnected = true;
  }


  byte cmd = parser.parse(); // pass it to the parser
  // parser returns non-zero when a pfod command is fully parsed
  if (cmd != 0) { // have parsed a complete msg { to }
    byte* pfodFirstArg = parser.getFirstArg(); // may point to \0 if no arguments in this msg.
    long pfodLongRtn; // used for parsing long return arguments, if any
    if ('.' == cmd) {
//     Serial.println("Got main menu requiest");
      // pfodApp has connected and sent {.} , it is asking for the main menu
      // send back the menu designed
      sendMainMenu();

      // now handle commands returned from button/sliders
    } else if ('A' == cmd) { // user moved slider -- 'Led is '
//     Serial.println("Got cmd A");
      // set output based on slider 0 == LOW, 1 == HIGH
      parser.parseLong(pfodFirstArg, &pfodLongRtn); // parse first arg as a long
      digitalWrite(cmd_A_pin, pfodLongRtn); // set output
      sendMainMenuUpdate(); // always send back a pfod msg otherwise pfodApp will disconnect.

    } else if ('!' == cmd) {
//     Serial.println("Got close cmd");
      // CloseConnection command
      closeConnection(parser.getPfodAppStream());
    } else {
      // unknown command
      parser.print(F("{}")); // always send back a pfod msg otherwise pfodApp will disconnect.
    }
  }
  //  <<<<<<<<<<<  Your other loop() code goes here

}

void closeConnection(Stream *io) {
  // add any special code here to force connection to be dropped
  parser.closeConnection(); // nulls io stream
  //alreadyConnected = false;
  // LBTServer.end();
  // startServer();
}

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() {
  Serial.println("send update");
  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("LinkItONE\nLed Control"));
  // send menu items
  parser.print(F("|A~Led is `"));
  Serial.println("befor readPin");
  parser.print(digitalRead(cmd_A_pin)); // read current output state 0 Low or 1 High
  Serial.println("after readPin");
  parser.print(F("~~Off\\On")); // 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 =========
```