Arduino MQTT Interface

Martin Leadbeater

January 28, 2013

Contents

Chapter 1

Introduction

"mquino.cpp" 5a ≡

```
    #include <Arduino.h>
    ⟨ declarations and functions 5b ⟩
    ⟨ the setup function 5c ⟩
    ⟨ the main loop function 8a ⟩
    ◇
```

We split the declarations section into smaller parts, taking care that everything will be presented to the compiler in the correct order.

⟨ declarations and functions 5b ⟩ ≡

```
    ⟨ include other headers and conditional code macros 5d, ... ⟩
    ⟨ constants and type definitions 6a, ... ⟩
    ⟨ shared class and structure definitions ? ⟩
    ⟨ classes and structures 9c ⟩
    ⟨ function declarations 11c, ... ⟩
    ⟨ global variables 6b, ... ⟩
    ⟨ function implementations 6d, ... ⟩
    ◇
```

Macro referenced in 5a, 9b.

The setup function is called early in the process of configuring the micro controller. It is defined as a simple `void` function.

⟨ the setup function 5c ⟩ ≡

```
    void setup() {
        ⟨ special microcontroller initialisation 8b ⟩
        ⟨ program initialisation steps 6c, ... ⟩
    }
    ◇
```

Macro referenced in 5a.

1.1 Program settings

The program loads its settings from an MQTT broker but needs to know where to find that broker. When the program starts, it uses DHCP to obtain a network address and loads the broker host name and port from EEPROM.

⟨ include other headers and conditional code macros 5d ⟩ ≡

```
    #include <EEPROM.h>
    ◇
```

Macro defined by 5d, 10cd.
Macro referenced in 5b.

We define a structure for permanent data and later, we provide some serial port commands to update this data from a PC connected via USB cable.

⟨ constants and type definitions 6a ⟩ ≡

```
struct ProgramSettings {
    byte header[2];
    char hostname[40];
    byte ip[4];
    byte mac_address[6];
    char broker_host[40];
    int broker_port;
    void load();
    void save();
    bool valid() { return header[0] == 217 && header[1] == 59; }
};
◇
```

Macro defined by 6a, 12b, 14b.
Macro referenced in 5b.

⟨ global variables 6b ⟩ ≡

```
ProgramSettings program_settings;
◇
```

Macro defined by 6b, 8c, 10e, 11d, 15a, 22c.
Macro referenced in 5b.

Data is stored in the EEPROM as a continuous block. We reserver address 0 for the our settings structure. At boot time, we load the program settings and we only trust them if the header is set correctly

⟨ program initialisation steps 6c ⟩ ≡

```
    program_settings.load();
    if (!program_settings.valid()) {
        program_settings.header[0] = 217;
        program_settings.header[1] = 59;
        strcpy(program_settings.broker_host,"0.0.0.0");
        strcpy(program_settings.hostname,"MyMega");
        for (byte i = 0; i<6; i++)
            program_settings.mac_address[i], MAC_ADDRESS[i];
        program_settings.save();
    }
◇
```

Macro defined by 6c, 8e, 11a, 12a.
Macro referenced in 5c.

⟨ function implementations 6d ⟩ ≡

```
void ProgramSettings::load() {
    int addr = 0;
    byte* p = (byte*)this;
    while (addr < sizeof(program_settings)) {
        *p++ = EEPROM.read(addr++);
    }
}
◇
```

Macro defined by 6d, 7ac, 9d, 10a, 12c, 20ac, 21b, 22a, 23ac.
Macro referenced in 5b.

⟨function implementations 7a⟩ ≡

```
void ProgramSettings::save() {
    int addr = 0;
    byte* p = (byte*)this;
    while (addr < sizeof(program_settings)) {
        EEPROM.write(addr++, *p++);
    }
}
◇
```

Macro defined by 6d, 7ac, 9d, 10a, 12c, 20ac, 21b, 22a, 23ac.
Macro referenced in 5b.

1.1.1 Test cases

⟨declare dummy version of necessary Arduino library symbols 7b⟩ ≡

```
typedef uint8_t byte;
◇
```

Macro defined by 7b, 23de, 24a.
Macro referenced in 9a.

⟨function implementations 7c⟩ ≡

```
#ifdef TESTING
class TestSettingsSave : public Test {
    int testNum;
    public:
        TestSettingsSave(int test) : testNum(test) {  }
        bool execute() {
            if (testNum == 1) return testOne();
        }

        bool testOne() {
            program_settings.header[0] = 217;
            program_settings.header[1] = 59;
            strcpy(program_settings.hostname, "TestOneHost");
            program_settings.broker_port = 5594;
            program_settings.save();
            program_settings.broker_port = 2225;
            strcpy(program_settings.hostname, "EMPTY");
            program_settings.load();
            if (program_settings.broker_port != 5594
                || strcmp(program_settings.hostname, "TestOneHost") != 0)
                return false;
            else
                return true;
        }
};
#endif
◇
```

Macro defined by 6d, 7ac, 9d, 10a, 12c, 20ac, 21b, 22a, 23ac.
Macro referenced in 5b.

⟨prepare test case 7d⟩ ≡

```
    TestSettingsSave testSaveSettings(1);
    Test::add(&testSaveSettings);
◇
```

Macro defined by 7d, 22b.
Macro referenced in 9b.

1.2 The main loop

⟨ the main loop function 8a ⟩ ≡

```
     void loop() {
         ⟨ declare local shared variables ? ⟩
         ⟨ check the connection and connect if necessary 11b ⟩
         ⟨ get the current time into variable ‘now’ 8d ⟩
         ⟨ protect against clock wrap-around ? ⟩
         ⟨ check and handle command input, return if necessary 15b ⟩
         ⟨ check inputs for change of state or publish timer and publish their status 14a ⟩
     }
     ◇
```

Macro referenced in 5a.

The program uses the serial port to receive local configuration parameters to simplify the problem of getting the program running without the usual network services such as DHCP etc.

⟨ special microcontroller initialisation 8b ⟩ ≡

```
         Serial.begin(115200);
     ◇
```

Macro referenced in 5c.

We load the program settings from EEPROM on startup and provide a way to update them via an MQTT channel and via the serial port.

1.3 Loop timer

⟨ global variables 8c ⟩ ≡

```
     unsigned long now;
     unsigned long publish_time;
     ◇
```

Macro defined by 6b, 8c, 10e, 11d, 15a, 22c.
Macro referenced in 5b.

⟨ get the current time into variable ‘now’ 8d ⟩ ≡

```
      now = millis(); ◇
```

Macro referenced in 8a.

⟨ program initialisation steps 8e ⟩ ≡

```
         now = millis();
         publish_time = now + 5000; // startup delay before we start publishing
     ◇
```

Macro defined by 6c, 8e, 11a, 12a.
Macro referenced in 5c.

Here we send data to the PC to be used for logging and also to display animated controls while the machine is being used.

1.4 Testing

Along with the program itself, we generate test cases and a test driver program. The outline of the test program is as follows. To enable the test routines to use exactly the same code as the program, we define some stub routines that simulate the arduino library functions. We define a symbol TESTING that we can use to indicate when code is only to be used for the test routines.

"arduino_stubs.h" 9a ≡

```
#include <iostream>
#define TESTING 1
⟨ declare dummy version of necessary Arduino library symbols 7b, ... ⟩
⟨ implement dummy version of necessary Arduino library symbols 24b, ... ⟩
◇
```

"test_driver.cpp" 9b ≡

```
#include "arduino_stubs.h"
#include <iostream>
#include <list>
⟨ declarations and functions 5b ⟩
int main(int argc, char *argv[]) {
    ⟨ prepare test case 7d, ... ⟩
    for (std::list<Test *>::iterator iter = Test::begin(); iter != Test::end(); iter++)
    {
        Test *test = *iter;
        test->run();
    }
    std::cout << Test::total() << " tests executed.\n"
              << Test::failures() << " failures\n"
              << Test::successes() << " passed\n";
    return 0;
}
◇
```

⟨ classes and structures 9c ⟩ ≡

```
#ifdef TESTING
class Test{
    public:
        void run();
        virtual bool execute() = 0;
        inline static std::list<Test *>::iterator begin() { return all_tests.begin(); }
        inline static std::list<Test *>::iterator end() { return all_tests.end(); }
        static void add(Test *test) { all_tests.push_back(test); }
        static int total() { return total_tests; }
        static int failures() { return total_failures; }
        static int successes() { return total_successes; }
    protected:
        static int total_tests;
        static int total_failures;
        static int total_successes;
    private:
        static std::list<Test *> all_tests;
};
#endif
◇
```

Macro referenced in 5b.

⟨ function implementations 9d ⟩ ≡

```
#ifdef TESTING
int Test::total_tests = 0;
int Test::total_failures = 0;
int Test::total_successes = 0;
std::list<Test *> Test::all_tests;
#endif
◇
```

Macro defined by 6d, 7ac, 9d, 10a, 12c, 20ac, 21b, 22a, 23ac.
Macro referenced in 5b.

⟨function implementations 10a⟩ ≡

```
#ifdef TESTING
void Test::run()
{
    ++total_tests;
    if (this->execute())
        ++total_successes;
    else
        ++total_failures;
}
#endif
◇
```

Macro defined by 6d, 7ac, 9d, 10a, 12c, 20ac, 21b, 22a, 23ac.
Macro referenced in 5b.

1.5 Debug Reporting

At the end of each loop, the program may return a standard report to the PC.

⟨generate report 10b⟩ ≡

```
#ifdef DEBUG

    Serial.print("\n");
#endif
◇
```

Macro never referenced.

This version of the program does not enable the DEBUG flag

⟨include other headers and conditional code macros 10c⟩ ≡

```
//#define DEBUG 1
◇
```

Macro defined by 5d, 10cd.
Macro referenced in 5b.

1.6 MQTT Interface

⟨include other headers and conditional code macros 10d⟩ ≡

```
#define USEMQTT 1
#ifdef USEMQTT
#include <SPI.h>
#include <PubSubClient.h>
#include <Ethernet.h>
#endif
◇
```

Macro defined by 5d, 10cd.
Macro referenced in 5b.

⟨global variables 10e⟩ ≡

```
uint16_t port = 1883;
byte MAC_ADDRESS[] = { 0x00, 0x01, 0x03, 0x41, 0x30, 0xA5 }; // old 3com card
#ifdef USEMQTT
char config_topic[30];
char message_buf[100];

EthernetClient enet_client;
PubSubClient client("127.0.0.1", 1883, callback, enet_client);
#endif
◇
```

Macro defined by 6b, 8c, 10e, 11d, 15a, 22c.
Macro referenced in 5b.

Initialise the ethernet MAC address and MQTT client.

⟨ program initialisation steps 11a ⟩ ≡

```
#ifdef USEMQTT

  if (Ethernet.begin(program_settings.mac_address) == 0)
  {
      Serial.println("Failed to configure Ethernet using DHCP");
      return;
  }
  client = PubSubClient(program_settings.hostname, program_settings.broker_port, callback, enet_client);
#endif
◇
```

Macro defined by 6c, 8e, 11a, 12a.
Macro referenced in 5c.

When we connect to the server, we subscribe to the configuration settings for the arduino.

⟨ check the connection and connect if necessary 11b ⟩ ≡

```
#ifdef USEMQTT
  if (!client.connected())
  {
      // clientID, username, MD5 encoded password
      client.connect("mquino", "mquino_user", "00000000000000000000000000000");
      snprintf(config_topic, 29, "%s/config/+", program_settings.hostname);
      client.subscribe(config_topic);
  }
#endif
◇
```

Macro referenced in 8a.

Subscribed data arrives via a callback

⟨ function declarations 11c ⟩ ≡

```
#ifdef USEMQTT
void callback(char* topic, byte* payload, unsigned int length);
#endif
◇
```

Macro defined by 11c, 19c, 20b, 21a, 22d, 23b.
Macro referenced in 5b.

The program expects messages in one of two formats:

- name '/' config '/' "dig" '/' pin_number ' ' ("IN" — "OUT" — "PWM")
- name '/' "dig" '/' pin_number ' ' ("on" — "off" — value)

where `value` is a number from 0 to 255, representing the duty cycle of the PWM.

The first format is used to configure ports of the arduino and the second is used to turn outputs on and off. In MQTT terms, the arduino will subscribe to the "OUT" and "PWM" topics and will publish changes on the "IN" topics.

⟨ global variables 11d ⟩ ≡

```
#ifdef USEMQTT
int pin_settings[64];
#endif
◇
```

Macro defined by 6b, 8c, 10e, 11d, 15a, 22c.
Macro referenced in 5b.

⟨ program initialisation steps 12a ⟩ ≡

```
#ifdef USEMQTT
    for(int i=0; i<64; ++i) pin_settings[i] = s_unknown;
#endif
◇
```

Macro defined by 6c, 8e, 11a, 12a.
Macro referenced in 5c.

⟨ constants and type definitions 12b ⟩ ≡

```
#ifdef USEMQTT
enum ParsingState { ps_unknown, ps_processing_config, ps_setting_output, ps_skipping };
enum Field { f_name, f_config, f_dig, f_pin, f_setting};
enum Setting { s_on, s_off, s_pwm, s_value, s_unknown, s_in, s_out };
#endif
◇
```

Macro defined by 6a, 12b, 14b.
Macro referenced in 5b.

⟨ function implementations 12c ⟩ ≡

```
#ifdef USEMQTT
void callback(char* topic, byte* payload, unsigned int length) {

  unsigned int i = 0;
  ParsingState parse_state = ps_unknown;
  int pin = -1;

  Serial.println("Message arrived:  topic: " + String(topic));
  Serial.println("Length: " + String(length,DEC));

  // create character buffer with ending null terminator (string)
  Field field = f_name;
  int j = 0;
  for(i=0; i<length; i++) {
    char curr = payload[i];
    if (curr == ’/’ || curr == ’ ’ || i + 1 == length) {
        message_buf[j] = 0;
        ⟨ process the current field 13 ⟩

        j = 0;
    }
    else {
            message_buf[j++] = curr;
    }
  }

  if (parse_state == ps_skipping)
    Serial.println(" parse error");
}
#endif
◇
```

Macro defined by 6d, 7ac, 9d, 10a, 12c, 20ac, 21b, 22a, 23ac.
Macro referenced in 5b.

⟨process the current field 13⟩ ≡

```
if (field == f_name) field = f_config; // ignore
else if (field == f_config) {
    if (strcmp(message_buf, "config") == 0) {
        parse_state = ps_processing_config;
        field = f_pin;
    }
    else if (strcmp(message_buf, "dig") == 0) {
        parse_state = ps_setting_output;
        field = f_pin; // already found f_dig
    }
    else {
        parse_state = ps_skipping;
        break;
    }
}
else if (field == f_dig) {
    if (strcmp(message_buf, "dig") == 0) {
        parse_state = ps_setting_output;
        field = f_pin; // found f_dig
    }
    else {
        parse_state = ps_skipping;
        break;
    }
}
else if (field == f_pin) {
    int pos = 0;
    pin = getNumber(message_buf, pos);
    if (pos == 0) {
        parse_state = ps_skipping;
        break;
    }
    field = f_setting;
}
else if (field == f_setting) {
    Setting setting = s_unknown;
    if (strcmp(message_buf, "IN")) setting = s_in;
    else if (strcmp(message_buf, "OUT")) setting = s_out;
    else if (strcmp(message_buf, "PWM")) setting = s_pwm;
    else if (strcmp(message_buf, "on")) setting = s_on;
    else if (strcmp(message_buf, "off")) setting = s_off;
    else {
        Serial.println ("unknown setting type");
        break;
    }
    if (parse_state == ps_processing_config) {
        if (setting == s_out) {
            pinMode(pin, OUTPUT);
            snprintf(message_buf, 99, "%s/pin/%d", program_settings.hostname, pin);
            client.subscribe(message_buf);
            if (pin < 64) pin_settings[pin] = s_in;
        }
        else if (setting == s_in) {
            pinMode(pin, INPUT);
            snprintf(message_buf, 99, "%s/pin/%d", program_settings.hostname, pin);
            const char *status = (digitalRead(pin)) ? "on" : "off";
            client.publish(message_buf, (uint8_t*)status, strlen(status), true );
            if (pin <64) pin_settings[pin] = s_in;
        }
        else if (setting == s_pwm) {
            Serial.println ("PWM mode is not currently supported");
        }
    }
    else if (parse_state == ps_setting_output) {
        if (setting == s_on)
            digitalWrite(pin, HIGH);
        else if (setting == s_off)
            digitalWrite(pin, LOW);
    }
```

Command	Parameters	Description
		Raw monitoring commands
Fn	none	Return the value (float) of analogue input number n where $0 <= n <= 5$
In	none	Return the value (H or L) of digital input number n where $0 <= n <= 63$
On	H or L	set the digital output n to High or Low where $0 <= n <= 63$. Using this function will automatically configure the port for output if necessary
		Program info and setting commands
?	none	return firmware id and version
s	none	save current volatile program settings to EEPROM
h	hostname	set the arduino host name (max 39 chars)
b	hostname	set the broker hostname
p	port	set the broker port number
d	none	display the current volatile settings
m	mac address	set the MAC address
i	ip address	set the default IP address

Table 1.1: Command Reference

1.6.1 Publishing updates

When the arduino is configured, it repeatedly publishes the status of its inputs to the MQTT broker. This version simply sends values every second. It needs to be upgraded to check more frequently for changes but still republish all entries frequently in case of packet loss.

⟨ check inputs for change of state or publish timer and publish their status 14a ⟩ ≡

```
#ifdef USEMQTT
    if (publish_time >= now) {
        for (byte i = 0; i<64; ++i) {
            if (pin_settings[i] == s_in) {
                snprintf(message_buf, 99, "%s/pin/%d", program_settings.hostname, i);
                const char *status = (digitalRead(i)) ? "on" : "off";
                client.publish(message_buf, (uint8_t*)status, strlen(status), true );
            }
        }
        publish_time += 1000;
    }
#endif
```
◇

Macro referenced in 8a.

1.7 Command processing

1.8 Input parser

The command protocol follows a request-response format, with requests and responses both beginning with a marker character, '>' and ending with a linefeed character. Neither marker are retained in the command itself. All data between the end marker and the begin marker are silently ignored.

⟨ constants and type definitions 14b ⟩ ≡

```
    enum InputStates{ idle, reading, command_loaded };
```
◇

Macro defined by 6a, 12b, 14b.
Macro referenced in 5b.

The input buffer is used for parsing commands on the serial port or messages from MQTT. The start, response and end mark characters are used for the serial port.

⟨ global variables 15a ⟩ ≡

```
const int INPUT_BUFSIZE = 60;
const int START_MARK = '>';
const int END_MARK = '\n';
const char *RESPONSE_START = "<";
InputStates input_state = idle;
char command[INPUT_BUFSIZE];
int input_pos = 0;
```

◇

Macro defined by 6b, 8c, 10e, 11d, 15a, 22c.
Macro referenced in 5b.

⟨ check and handle command input, return if necessary 15b ⟩ ≡

```
        bool response_required = false;
        const char *error_message = 0;
        int chars_ready = Serial.available();
        if (input_state != command_loaded && chars_ready) {
            ⟨ process serial input 16a ⟩
        }
        else if (input_state == command_loaded) {
#ifdef DEBUG
            Serial.println("command loaded");
#endif
            response_required = true;
            char cmd = command[0];
            int scan = 1;
            int param1 = getNumber(command, scan); // read number from this index
            int param2 = getNumber(command, scan); // read the paramer
            int paramLen = getString(command, scan);
            if (cmd == '?') {  ⟨ process enquiry command 16b ⟩ }
            else if (cmd == 'd') { ⟨ process display command 18c ⟩ }
            else if (cmd == 'h') { ⟨ process host command 17a ⟩ }
            else if (cmd == 'b') { ⟨ process broker command 17b ⟩ }
            else if (cmd == 'p') { ⟨ process port command 18a ⟩ }
            else if (cmd == 's') { ⟨ process save command 18b ⟩ }
            else if (cmd == 'm') { ⟨ process mac address command 17c ⟩ }
            else if (cmd == 'i') { ⟨ process ip address command 17d ⟩ }
            else if (cmd == 'F') { ⟨ process analogue input command 19a ⟩ }
            else if (cmd == 'I') {  ⟨ process digital input command 18d ⟩ }
            else if (cmd == 'O') {  ⟨ process digital output command 19b ⟩ }
        done_command:
        // remove the command from the input buffer
        char *p = command;
        char *q = command + input_pos;
        while (*q) {
            *p++ = *q++;
        }
        *p = 0;
        input_pos = p - command;
        input_state = idle;
        if (error_message) {
            Serial.print(RESPONSE_START);
            Serial.println(error_message);
        }
        else if (response_required) {
            Serial.print(RESPONSE_START);
            Serial.println("OK");
        }
    }
```

◇

Macro referenced in 8a.

⟨ process serial input 16a ⟩ ≡

```
    int ch = Serial.read();
#ifdef DEBUG
    Serial.println(ch);
#endif
    switch (input_state) {
        case idle:
            if (ch == START_MARK) {
                input_state = reading;
#ifdef DEBUG
                Serial.print("reading (");
                Serial.print(chars_ready);
                Serial.println(")");
#endif
            }
            break;
        case reading:
            if (ch == END_MARK) {
#ifdef DEBUG
            Serial.println("end mark");
#endif
                if (input_pos == 0) {
                    input_state = idle; // no command read
#ifdef DEBUG
                    Serial.println("idle");
#endif
                }
                else {
                    input_state = command_loaded;
#ifdef DEBUG
                    Serial.println("loaded");
#endif
                }
#ifdef DEBUG
                Serial.print("buf: ");
                Serial.println(command);
#endif
                break;
            }
            command[input_pos++] = ch;
            if (input_pos >= INPUT_BUFSIZE) // buffer overrun
            {
                input_state = idle;
                input_pos = 0;
            }
            command[input_pos] = 0; // keep the input string terminated
            break;
        case command_loaded:
            break;
         default: ;
    }
```

◇

Macro referenced in 15b.

1.8.1 Command handlers

⟨ process enquiry command 16b ⟩ ≡

```
    Serial.print(RESPONSE_START);
    Serial.println("mquino v0.2 Jan 28, 2013");
```

◇

Macro referenced in 15b.

⟨ process host command 17a ⟩ ≡

```
scan = 1;
paramLen = getString(command, scan);
if (paramLen < 40) {
    strcpy(program_settings.hostname, paramString);
    Serial.print("hostname set to ");
    Serial.println(paramString);
}
```

◇

Macro referenced in 15b.

⟨ process broker command 17b ⟩ ≡

```
scan = 1;
paramLen = getString(command, scan);
if (paramLen < 40) {
    strcpy(program_settings.broker_host, paramString);
    Serial.print("broker host set to ");
    Serial.println(paramString);
}
```

◇

Macro referenced in 15b.

⟨ process mac address command 17c ⟩ ≡

```
scan = 1;
int i = 0;
while (i<6 && command[scan] != 0) {
    program_settings.mac_address[i] = getHexNumber(command, scan);
    if (command[scan] == 0) break;
    ++scan;
    ++i;
}
Serial.print("MAC address is now: ");
for (int i=0; i<6; ++i) {
    if (program_settings.mac_address[i] < 10)
        Serial.print(’0’);
    Serial.print(program_settings.mac_address[i], HEX);
    if (i<5) Serial.print(’:’);
}
Serial.println();
```

◇

Macro referenced in 15b.

⟨ process ip address command 17d ⟩ ≡

```
scan = 1;
int i = 0;
while (i<4 && command[scan] != 0) {
    program_settings.ip[i] = getNumber(command, scan);
    if (command[scan] == 0) break;
    ++scan;
    ++i;
}
Serial.print("IP address is now: ");
for (int i=0; i<4; ++i) {
    Serial.print(program_settings.ip[i], DEC);
    if (i<3) Serial.print(’.’);
}
Serial.println();
```

◇

Macro referenced in 15b.

⟨ process port command 18a ⟩ ≡

```
        program_settings.broker_port = param1;
        Serial.print("port set to ");
        Serial.println(param1);
    ◇
```

Macro referenced in 15b.

⟨ process save command 18b ⟩ ≡

```
        scan = 1;
        program_settings.save();
    ◇
```

Macro referenced in 15b.

⟨ process display command 18c ⟩ ≡

```
        Serial.print("host      : "); Serial.println(program_settings.hostname);
        Serial.print("default ip: ");
        for (byte i=0; i<4; ++i) {
            Serial.print(program_settings.ip[i], DEC);
            if (i<3) Serial.print('.');
        }
        Serial.println();
        Serial.print("broker    : "); Serial.println(program_settings.broker_host);
        Serial.print("port      : "); Serial.println(program_settings.broker_port);
        Serial.print("mac       : ");
        for (byte i=0; i<6; ++i) {
            if (program_settings.mac_address[i] < 10)
                Serial.print('0');
            Serial.print(program_settings.mac_address[i], HEX);
            if (i<5) Serial.print(':');
        }
        Serial.println();
    #ifdef USEMQTT
        Serial.print("current ip: ");
        for (byte i = 0; i < 4; i++) {
          Serial.print(Ethernet.localIP()[i], DEC);
          if (i<3) Serial.print(".");
        }
    #endif
        Serial.println();
    ◇
```

Macro referenced in 15b.

Read a digital input and return H/L, depending on the result. If an invalid port is supplied, generate an error message;'

⟨ process digital input command 18d ⟩ ≡

```
    if (param1 >= 0 && param1 <= 64) {
        Serial.print(RESPONSE_START);
        if (digitalRead(param1))
            Serial.println("H");
        else
            Serial.println("L");
    }
    else
        error_message = "invalid port";
    ◇
```

Macro referenced in 15b.

⟨ process analogue input command 19a ⟩ ≡

```
        if (param1 >= 0 && param1 <= 5) {
            if (param1 == 0) param1 = A0;
            else if (param1 == 1) param1 = A1;
            else if (param1 == 2) param1 = A2;
            else if (param1 == 3) param1 = A3;
            else if (param1 == 4) param1 = A4;
            else if (param1 == 5) param1 = A5;
            else param1 = -1;
            if (param1 >= 0) {
                Serial.print(RESPONSE_START);
                Serial.println( analogRead( param1 ) );
            }
        }
        else
            error_message = "Analogue reads are only available for ports 0..5";
```

◇

Macro referenced in 15b.

⟨ process digital output command 19b ⟩ ≡

```
        if (param1 >= 0 && param1 <= 64 && paramLen == 1)
            if (paramString[0] == 'H')
                digitalWrite(param1, HIGH);
            else if (paramString[0] == 'L')
                digitalWrite(param1, LOW);
            else
                error_message = "bad output state";
        else
            error_message = "invalid port";
```

◇

Macro referenced in 15b.

1.8.2 Reading a number from the PC

When reading a number, leading spaces are skipped, the offset is updated to point to the first non numeric character after the leading spaces.

⟨ function declarations 19c ⟩ ≡

```
        int getNumber(char *buf_start, int &offset);
```

◇

Macro defined by 11c, 19c, 20b, 21a, 22d, 23b.
Macro referenced in 5b.

When parsing numbers we rely on the fact that the command buffer is always null terminated

⟨ function implementations 20a ⟩ ≡

```
int getNumber(char *buf_start, int &offset)
{
    char *p = buf_start + offset;
    int res = 0;
    while (*p == ' ') {  ++offset; p++; }
    int ch = *p;
    while (ch >= '0' && ch <= '9') {
        res = res * 10 + (ch - '0');
        ++offset;
        p++;
        ch = *p;
    }
    return res;
}
```

◇

Macro defined by 6d, 7ac, 9d, 10a, 12c, 20ac, 21b, 22a, 23ac.
Macro referenced in 5b.

When reading a hex number, leading spaces are skipped, the offset is updated to point to the first non numeric character after the leading spaces.

⟨ function declarations 20b ⟩ ≡

```
    int getHexNumber(char *buf_start, int &offset);
```

◇

Macro defined by 11c, 19c, 20b, 21a, 22d, 23b.
Macro referenced in 5b.

When parsing numbers we rely on the fact that the command buffer is always null terminated

⟨ function implementations 20c ⟩ ≡

```
char upper(char ch) {
    if (ch>='a' && ch<='z') ch = ch - 'a' + 'A';
    return ch;
}
int getHexNumber(char *buf_start, int &offset)
{
    char *p = buf_start + offset;
    int res = 0;
    while (*p == ' ') {  ++offset; p++; }
    int ch = upper(*p);
    while ( (ch >= '0' && ch <= '9') || (ch >= 'A' && ch <='F')) {
        res = res * 16;
        if (ch <= '9')
            res = res + (ch - '0');
        else
            res = res + (ch - 'A') + 10;
#ifdef DEBUG
        Serial.print("hex: ");
        Serial.print(res);
        Serial.print(" ");
#endif
        ++offset;
        p++;
        ch = upper(*p);
    }
    return res;
}
```

◇

Macro defined by 6d, 7ac, 9d, 10a, 12c, 20ac, 21b, 22a, 23ac.
Macro referenced in 5b.

⟨ function declarations 21a ⟩ ≡

```
        float getFloat(char *buf_start, int &offset);
```
◇

Macro defined by 11c, 19c, 20b, 21a, 22d, 23b.
Macro referenced in 5b.

As above, we rely on the fact that the command buffer is always null terminated.

⟨ function implementations 21b ⟩ ≡

```
    float getFloat(char *buf_start, int &offset)
    {
        bool seenDecimalPoint = false;
        char *p = buf_start + offset;
        float res = 0.0f;
        float frac = 1.0f;
        while (*p == ' ') {  ++offset; p++; }
        int ch = *p;
        while ( (ch >= '0' && ch <= '9') || (ch == '.' && !seenDecimalPoint) ) {
            if (ch == '.')
                seenDecimalPoint = true;
            else {
                int val = ch - '0';
                if (!seenDecimalPoint)
                    res = res * 10.0 + (float)val;
                else {
                    frac = frac/10.0f;
                    res = res + frac * val;
                }
            }
            ++offset;
            p++;
            ch = *p;
        }
        return res;
    }
```
◇

Macro defined by 6d, 7ac, 9d, 10a, 12c, 20ac, 21b, 22a, 23ac.
Macro referenced in 5b.

Test cases

⟨ function implementations 22a ⟩ ≡

```
#ifdef TESTING
class TestGetFloat : public Test {
    int testNum;
    public:
        TestGetFloat(short test) : testNum(test) {  }
        bool execute() {
            if (testNum == 1) return testOne();
        }

        bool testOne() {
            strcpy(command, "z 123.546 X");
            int offset = 1;
            float val = getFloat(command, offset);
            if (val == 123.546f)
                return true;
            else {
                std::cout << "Error, expected " << 123.546 << " got " << val << "\n";
                return false;
            }
        }
};
#endif
◇
```

Macro defined by 6d, 7ac, 9d, 10a, 12c, 20ac, 21b, 22a, 23ac.
Macro referenced in 5b.

⟨ prepare test case 22b ⟩ ≡

```
    TestGetFloat testGetFloat(1);
    Test::add(&testGetFloat);
◇
```

Macro defined by 7d, 22b.
Macro referenced in 9b.

1.8.3 Reading a string from the PC

⟨ global variables 22c ⟩ ≡

```
char paramString[40];
◇
```

Macro defined by 6b, 8c, 10e, 11d, 15a, 22c.
Macro referenced in 5b.

Define a function to get a string parameter. `getString\` returns the string length.

⟨ function declarations 22d ⟩ ≡

```
int getString(char *buf_start, int &offset);
◇
```

Macro defined by 11c, 19c, 20b, 21a, 22d, 23b.
Macro referenced in 5b.

⟨ function implementations 23a ⟩ ≡

```
int getString(char *buf_start, int &offset)
{
    char *p = buf_start + offset;
    while (*p == ' ') {  ++offset; p++; } // skip leading spaces
    char *q = paramString;
    while (q - paramString < 39 && *p && *p != ' ') {
        *q++ = *p++;
    }
    *q = 0;
    return q - paramString;
}
◇
```

Macro defined by 6d, 7ac, 9d, 10a, 12c, 20ac, 21b, 22a, 23ac.
Macro referenced in 5b.

1.9 Utility functions

⟨ function declarations 23b ⟩ ≡

```
bool opposite(float a, float b);
◇
```

Macro defined by 11c, 19c, 20b, 21a, 22d, 23b.
Macro referenced in 5b.

⟨ function implementations 23c ⟩ ≡

```
bool opposite(float a, float b)
{
  if (a<0 && b>0) return true;
  if (a>0 && b<0) return true;
  return false;
}
◇
```

Macro defined by 6d, 7ac, 9d, 10a, 12c, 20ac, 21b, 22a, 23ac.
Macro referenced in 5b.

1.10 Test Functions

⟨ declare dummy version of necessary Arduino library symbols 23d ⟩ ≡

```
#define OUTPUT 1
#define LOW 0
#define HIGH 1
◇
```

Macro defined by 7b, 23de, 24a.
Macro referenced in 9a.

⟨ declare dummy version of necessary Arduino library symbols 23e ⟩ ≡

```
struct SimulatedSerialPort {
    void print(int);
    void println(int);
    void print(float, int);
    void println(float, int);
    void print(const char *);
    void println(const char *);
};
◇
```

Macro defined by 7b, 23de, 24a.
Macro referenced in 9a.

⟨ declare dummy version of necessary Arduino library symbols 24a ⟩ ≡

```
void pinMode(int, int);
int analogRead(int);
void analogWrite(int, int);
void digitalWrite(int, int);
void delayMicroseconds(int);
void delay(int);
SimulatedSerialPort Serial;
◇
```

Macro defined by 7b, 23de, 24a.
Macro referenced in 9a.

⟨ implement dummy version of necessary Arduino library symbols 24b ⟩ ≡

```
void pinMode(int, int) {}
int analogRead(int) { return 0;}
void analogWrite(int, int) {}
void digitalWrite(int, int) {}
void delayMicroseconds(int) {}
void delay(int) {}
◇
```

Macro defined by 24bc.
Macro referenced in 9a.

⟨ implement dummy version of necessary Arduino library symbols 24c ⟩ ≡

```
    void SimulatedSerialPort::print(int a) { std::cout << a; }
    void SimulatedSerialPort::println(int a) {  std::cout << a << "\n"; }
    void SimulatedSerialPort::print(float a , int b) {  std::cout << a; }
    void SimulatedSerialPort::println(float a, int b) {  std::cout << a << "\n"; }
    void SimulatedSerialPort::print(const char *s) {  std::cout << s; }
    void SimulatedSerialPort::println(const char *s) {  std::cout << s << "\n"; }
◇
```

Macro defined by 24bc.
Macro referenced in 9a.

Chapter 2

Installation

2.1 Generating the program from the source file

⟨ compile the document using nuweb 25 ⟩ ≡

```
fname=‘basename "$1" .w‘
rm -f $fname.pdf
/usr/local/bin/nuweb $fname.w && pdflatex $fname.tex
if [ $? -eq 0 ]; then
    bibtex $fname
    [ -r $fname.idx ] && makeindex $fname
    pdflatex  $fname.tex
    pdflatex  $fname.tex
    [ -r $fname.pdf ] && open $fname.pdf
fi
◇
```

Macro never referenced.

Appendix A

Files

"arduino_stubs.h" Defined by 9a.
"mquino.cpp" Defined by 5a.
"test_driver.cpp" Defined by 9b.

Appendix B

Macros

⟨ check and handle command input, return if necessary 15b ⟩ Referenced in 8a.
⟨ check inputs for change of state or publish timer and publish their status 14a ⟩ Referenced in 8a.
⟨ check the connection and connect if necessary 11b ⟩ Referenced in 8a.
⟨ classes and structures 9c ⟩ Referenced in 5b.
⟨ compile the document using nuweb 25 ⟩ Not referenced.
⟨ constants and type definitions 6a, 12b, 14b ⟩ Referenced in 5b.
⟨ declarations and functions 5b ⟩ Referenced in 5a, 9b.
⟨ declare dummy version of necessary Arduino library symbols 7b, 23de, 24a ⟩ Referenced in 9a.
⟨ declare local shared variables ? ⟩ Referenced in 8a.
⟨ function declarations 11c, 19c, 20b, 21a, 22d, 23b ⟩ Referenced in 5b.
⟨ function implementations 6d, 7ac, 9d, 10a, 12c, 20ac, 21b, 22a, 23ac ⟩ Referenced in 5b.
⟨ generate report 10b ⟩ Not referenced.
⟨ get the current time into variable 'now' 8d ⟩ Referenced in 8a.
⟨ global variables 6b, 8c, 10e, 11d, 15a, 22c ⟩ Referenced in 5b.
⟨ implement dummy version of necessary Arduino library symbols 24bc ⟩ Referenced in 9a.
⟨ include other headers and conditional code macros 5d, 10cd ⟩ Referenced in 5b.
⟨ prepare test case 7d, 22b ⟩ Referenced in 9b.
⟨ process analogue input command 19a ⟩ Referenced in 15b.
⟨ process broker command 17b ⟩ Referenced in 15b.
⟨ process digital input command 18d ⟩ Referenced in 15b.
⟨ process digital output command 19b ⟩ Referenced in 15b.
⟨ process display command 18c ⟩ Referenced in 15b.
⟨ process enquiry command 16b ⟩ Referenced in 15b.
⟨ process host command 17a ⟩ Referenced in 15b.
⟨ process ip address command 17d ⟩ Referenced in 15b.
⟨ process mac address command 17c ⟩ Referenced in 15b.
⟨ process port command 18a ⟩ Referenced in 15b.
⟨ process save command 18b ⟩ Referenced in 15b.
⟨ process serial input 16a ⟩ Referenced in 15b.
⟨ process the current field 13 ⟩ Referenced in 12c.
⟨ program initialisation steps 6c, 8e, 11a, 12a ⟩ Referenced in 5c.
⟨ protect against clock wrap-around ? ⟩ Referenced in 8a.
⟨ shared class and structure definitions ? ⟩ Referenced in 5b.
⟨ special microcontroller initialisation 8b ⟩ Referenced in 5c.
⟨ the main loop function 8a ⟩ Referenced in 5a.
⟨ the setup function 5c ⟩ Referenced in 5a.

Appendix C

Identifiers

Bibliography