Arduino MQTT Interface

Martin Leadbeater

January 30, 2013

Contents

Chapter 1

Introduction

This is a program for the Arduino (http://arduino.cc) that uses MQTT (http://mqtt.org) to provide a general purpose hobbyist I/O platform. The program provides the following features:

- configuration of I/O pins via MQTT
- a serial (USB) interface for configuration of the Arduino
- support for digital input or output

The program requires an Ethernet shield or a builtin Ethernet device such as an EtherTen or EtherMega (http://freetronics.com/).

The main program is defined here, note that most Arduino 'sketches' do not require the include for Arduino.h but we use it to simplify commandline compilation.

```
"mquino.cpp" 5a ≡
    #include <Arduino.h>
    ⟨declarations and functions 5b⟩
    ⟨the setup function 5c⟩
    ⟨the main loop function 9a⟩
    ◇
```

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 10c⟩
    ⟨function declarations 6c, ... ⟩
    ⟨declare global variables that must be declared first 7b⟩
    ⟨global variables 9c, ... ⟩
    ⟨function implementations 7a, ... ⟩
    ◇
```

Macro referenced in 5a, 10b.

Arduino programs required two functions: `setup` and `loop`. The setup function is called early in the process of configuring the micro controller. It is defined as a simple `void` function. The `loop` function is described later, in Section 1.3

```
⟨the setup function 5c⟩ ≡

    void setup() {
        ⟨special microcontroller initialisation 9b⟩
        ⟨program initialisation steps 7c, ... ⟩
    }
    ◇
```

Macro referenced in 5a.

Command	**P**arameters	**D**escription
		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

1.1 Command processing

When connecting to the USB serial device via a terminal, the user can view and update settings on the Arduino. This is necessary for configuration of the device for MQTT communication.

1.2 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, 6b, 11cd.
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.

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.

⟨ 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;
    IPAddress broker_ip;
    IPAddress dns_address;
    IPAddress broker_address;
    void load();
    void save();
    bool valid() { return header[0] == 217 && header[1] == 59; }
    ProgramSettings(EthernetClient &);
};
◇
```

Macro defined by 6a, 13e, 18a.
Macro referenced in 5b.

The initialisation requires a host lookup

⟨ include other headers and conditional code macros 6b ⟩ ≡

```
#include <Dns.h>
◇
```

Macro defined by 5d, 6b, 11cd.
Macro referenced in 5b.

We provide a function to convert an ip number in a string to an array of bytes.

⟨ function declarations 6c ⟩ ≡

```
bool mq_inet_aton(const char *ipstring, byte *addr);
◇
```

Macro defined by 6c, 13b, 25b, 26ac, 28c, 29a.
Macro referenced in 5b.

Note that we assume the broker host is null terminated.

⟨ function implementations 7a ⟩ ≡

```
ProgramSettings::ProgramSettings(EthernetClient &enet_client) {
    load();
    if (!program_settings.valid()) {
        program_settings.header[0] = 217;
        program_settings.header[1] = 59;
        strcpy(program_settings.broker_host,"192.168.2.1");
        strcpy(program_settings.hostname,"MyMega");
        for (byte i = 0; i<6; i++)
            program_settings.mac_address[i] = MAC_ADDRESS[i];
        // setup the broker up address default (ethernet is not available yet)
        broker_ip[0] = 192; broker_ip[1] = 168; broker_ip[2] = 2; broker_ip[3] = 1;
        program_settings.save();
    }
    Ethernet.begin(mac_address);
    DNSClient dns;
    dns.begin(dns_address);
    dns.getHostByName(broker_host, broker_ip);
}
◇
```

Macro defined by 7ade, 8ac, 11ab, 14a, 16b, 25c, 26b, 27ab, 28d, 29b.
Macro referenced in 5b.

Since we want to initialise the Ethernet client and the MQTT publisher/subcriber client using the settings loaded from the EEPROM, we define the `program_settings` variable at the top of the globals. The constructor will automatically call `load()` to initialise the structure.

⟨ declare global variables that must be declared first 7b ⟩ ≡

```
EthernetClient enet_client;
ProgramSettings program_settings(enet_client);
◇
```

Macro referenced in 5b.

⟨ program initialisation steps 7c ⟩ ≡

```

◇
```

Macro defined by 7c, 9e, 12c, 13d.
Macro referenced in 5c.

⟨ function implementations 7d ⟩ ≡

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

Macro defined by 7ade, 8ac, 11ab, 14a, 16b, 25c, 26b, 27ab, 28d, 29b.
Macro referenced in 5b.

⟨ function implementations 7e ⟩ ≡

```
bool mq_inet_aton(const char *ipstring, IPAddress &addr) {
    return false;
}
◇
```

Macro defined by 7ade, 8ac, 11ab, 14a, 16b, 25c, 26b, 27ab, 28d, 29b.
Macro referenced in 5b.

⟨ function implementations 8a ⟩ ≡

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

Macro defined by 7ade, 8ac, 11ab, 14a, 16b, 25c, 26b, 27ab, 28d, 29b.
Macro referenced in 5b.

1.2.1 Test cases

⟨ declare dummy version of necessary Arduino library symbols 8b ⟩ ≡

```
typedef uint8_t byte;
◇
```

Macro defined by 8b, 17c, 29cd, 30ab.
Macro referenced in 10a.

⟨ function implementations 8c ⟩ ≡

```
#ifdef TESTING
class TestSettingsSave : public Test {
    int testNum;
    public:
        TestSettingsSave(int test) : Test("Test Settings Save", ""), 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 7ade, 8ac, 11ab, 14a, 16b, 25c, 26b, 27ab, 28d, 29b.
Macro referenced in 5b.

⟨ prepare test case 8d ⟩ ≡

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

◇

Macro defined by 8d, 17b, 28a.
Macro referenced in 10b.

1.3 The main loop

⟨ the main loop function 9a ⟩ ≡

```
void loop() {
    ⟨ declare local shared variables ? ⟩
    ⟨ check the connection and connect if necessary 12d ⟩
    ⟨ poll MQTT 13a ⟩
    ⟨ get the current time into variable ‘now’ 9d ⟩
    ⟨ protect against clock wrap-around ? ⟩
    ⟨ check and handle command input, return if necessary 19 ⟩
    ⟨ check inputs for change of state or publish timer and publish their status 17d ⟩
}
```

◇

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 9b ⟩ ≡

```
    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.

1.4 Loop timer

⟨ global variables 9c ⟩ ≡

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

Macro defined by 9c, 12ab, 13c, 18b, 28b.
Macro referenced in 5b.

⟨ get the current time into variable ‘now’ 9d ⟩ ≡

```
 now = millis(); ◇
```

Macro referenced in 9a.

⟨ program initialisation steps 9e ⟩ ≡

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

Macro defined by 7c, 9e, 12c, 13d.
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.

1.5 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"` 10a ≡

```
#include <iostream>
#define TESTING 1
⟨ declare dummy version of necessary Arduino library symbols 8b, ... ⟩
⟨ implement dummy version of necessary Arduino library symbols 30c, ... ⟩
◇
```

"test_driver.cpp" 10b ≡

```
#include "arduino_stubs.h"
#include <iostream>
#include <list>
⟨ declarations and functions 5b ⟩
int main(int argc, char *argv[]) {
    ⟨ prepare test case 8d, ... ⟩
    for (std::list<Test *>::iterator iter = Test::begin(); iter != Test::end(); iter++)
    {
        Test *test = *iter;
        std::cout << test->getName();
        if (test->getDesc().length())
            std::cout << "(" << test->getDesc() << ")";
        std::cout << ": ";
        if (test->run())
            std::cout << "passed\n";
        else
            std::cout << "failed\n";
    }
    std::cout << Test::total() << " tests executed.\n"
              << Test::failures() << " failures\n"
              << Test::successes() << " passed\n";
    return 0;
}
◇
```

⟨ classes and structures 10c ⟩ ≡

```
#ifdef TESTING
class Test{
    public:
        Test(const char *test_name, const char *test_desc) : name(test_name), description(test_desc) {}
        bool 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; }
        const std::string & getName() const { return name; }
        const std::string & getDesc() const { return description; }
    protected:
        std::string name;
        std::string description;
        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 11a⟩ ≡

```
#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 7ade, 8ac, 11ab, 14a, 16b, 25c, 26b, 27ab, 28d, 29b.
Macro referenced in 5b.

⟨function implementations 11b⟩ ≡

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

Macro defined by 7ade, 8ac, 11ab, 14a, 16b, 25c, 26b, 27ab, 28d, 29b.
Macro referenced in 5b.

1.6 Debug Reporting

The program uses several debug flags to assist with diagnosing isses:

DEBUG_CONSOLE display information about parsing user input on the console

DEBUG display general debug messages

FEEDBACK display feedback about the MQTT channel

This version of the program does not enable the DEBUG flag.

⟨include other headers and conditional code macros 11c⟩ ≡

```
//#define DEBUG_CONSOLE
//#define DEBUG 1
#define FEEDBACK
◇
```

Macro defined by 5d, 6b, 11cd.
Macro referenced in 5b.

1.7 MQTT Interface

⟨include other headers and conditional code macros 11d⟩ ≡

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

Macro defined by 5d, 6b, 11cd.
Macro referenced in 5b.

⟨ global variables 12a ⟩ ≡

```
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];
#endif
◇
```

Macro defined by 9c, 12ab, 13c, 18b, 28b.
Macro referenced in 5b.

The ethernet client is initialised in the setup function but at present, we have not found a reliable way to have the PubSubClient object initialise within the `setup` method.

⟨ global variables 12b ⟩ ≡

```
byte server[] = { 192, 168, 2, 1 };

PubSubClient client(server, 1883, callback, enet_client);
◇
```

Macro defined by 9c, 12ab, 13c, 18b, 28b.
Macro referenced in 5b.

Initialise the ethernet MAC address and MQTT client.

⟨ program initialisation steps 12c ⟩ ≡

```
#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 7c, 9e, 12c, 13d.
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 12d ⟩ ≡

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

Macro referenced in 9a.

⟨ poll MQTT 13a ⟩ ≡

```
    client.loop();
◇
```

Macro referenced in 9a.

Subscribed data arrives via a callback

topic	*message*
name '/' "config" '/' "dig" '/' pin_number	"IN" or "OUT" or "PWM"
name '/' "dig" '/' "pin_number"	"on" or "off" or value

Table 1.2: Expected message formats

⟨ function declarations 13b ⟩ ≡

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

Macro defined by 6c, 13b, 25b, 26ac, 28c, 29a.
Macro referenced in 5b.

The program expects messages in one of two formats as shown in Figure 1.2.
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 13c ⟩ ≡

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

Macro defined by 9c, 12ab, 13c, 18b, 28b.
Macro referenced in 5b.

⟨ program initialisation steps 13d ⟩ ≡

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

Macro defined by 7c, 9e, 12c, 13d.
Macro referenced in 5c.

⟨ constants and type definitions 13e ⟩ ≡

```
#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, 13e, 18a.
Macro referenced in 5b.

The `callback` method is called whenever a message arrives from MQTT.

⟨function implementations 14a⟩ ≡

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

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

  Serial.print("Message arrived\n  topic: ");
  Serial.println(topic);
  Serial.print("Message length: ");
  Serial.println(length);

  Field field = f_name;
  int j = 0;
  unsigned int n = strlen(topic);
  for(i=0; i<=n; i++) {
    char curr = (i<n) ? topic[i] : 0;
    if (curr == '/' || curr == ' ' || i == n ) {
        message_buf[j] = 0;
        ⟨process the current field 14b⟩

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

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

Macro defined by 7ade, 8ac, 11ab, 14a, 16b, 25c, 26b, 27ab, 28d, 29b.
Macro referenced in 5b.

⟨ process the current field 14b ⟩ ≡

```
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_dig;
    }
    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) {
        if (parse_state == ps_unknown) 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;

    Setting setting = s_unknown;
    if (strncmp((const char *)payload, "IN", length) == 0) setting = s_in;
    else if (strncmp((const char *)payload, "OUT", length) == 0) setting = s_out;
    else if (strncmp((const char *)payload, "PWM", length) == 0) setting = s_pwm;
    else if (strncmp((const char *)payload, "ON", length) == 0) setting = s_on;
    else if (strncmp((const char *)payload, "OFF", length) == 0) setting = s_off;
    else {
        Serial.println ("unknown setting type");
        break;
    }
    if (parse_state == ps_processing_config) {
        if (setting == s_out) {
            ⟨ subscribe to the topic that indicates changes on an output pin 15 ⟩
        }
        else if (setting == s_in) {
            ⟨ publish changes on an input pin 16a ⟩
        }
        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);
            Serial.print("turned pin "); Serial.print(pin); Serial.println(" on");
        }
        else if (setting == s_off) {
            digitalWrite(pin, LOW);
            Serial.print("turned pin "); Serial.print(pin); Serial.println(" off");
        }
    }
    break;
}
```
◇

Macro referenced in 14

The topic for an arduino input pin is *controller-name*/dig/*pin-number*. If we have been asked to configure a pin that is out of range, we do nothing. This scrap needs more work to cater or different hardware features.

⟨ subscribe to the topic that indicates changes on an output pin 15 ⟩ ≡

```
if (pin < 64) {
    pinMode(pin, OUTPUT);
    pin_settings[pin] = s_out;
    snprintf(message_buf, 99, "%s/dig/%d", program_settings.hostname, pin);
#ifdef FEEDBACK
    Serial.print("subscribing to: ");
    Serial.println(message_buf);
#endif
    client.subscribe(message_buf);
}
◇
```

Macro referenced in 14b.

⟨ publish changes on an input pin 16a ⟩ ≡

```
if (pin <64) {
    pinMode(pin, INPUT);
    pin_settings[pin] = s_in;
    snprintf(message_buf, 99, "%s/dig/%d", program_settings.hostname, pin);
    const char *status = (digitalRead(pin)) ? "on" : "off";
#ifdef FEEDBACK
    Serial.print("publishing to: ");
    Serial.println(message_buf);
#endif
    client.publish(message_buf, (uint8_t*)status, strlen(status), true );
}
◇
```

Macro referenced in 14b.

1.7.1 Test cases

⟨ function implementations 16b ⟩ ≡

```
#ifdef TESTING

class TestCallback : public Test {
    int testNum;
    public:
        TestCallback(short test) : Test("Test callback function", ""), testNum(test) {  }
        bool execute() {
            if (testNum == 1) return testOne();
            else if (testNum == 2) return testTwo();
        }

        ⟨ implement a callback test for configuration of a digital input 16c ⟩
        ⟨ implement a callback test for configuration of a digital output 17a ⟩
};
#endif
◇
```

Macro defined by 7ade, 8ac, 11ab, 14a, 16b, 25c, 26b, 27ab, 28d, 29b.
Macro referenced in 5b.

In this test, we call the callback function with a path to a single digital pin and set the message to ”IN”. Note that we add extra data in the messge buffer since that field is really a byte array and the callback function should use the length as given and not use `strlen`.

⟨ implement a callback test for configuration of a digital input 16c ⟩ ≡

```
bool testOne() {
    description = "configure a digital input";
    pin_settings[5] == s_unknown;
    char *topic = strdup("MyMega/config/dig/5");
    callback(topic, (byte*)"INxx", 2);
    free(topic);
    if (pin_settings[5] == s_in) return true;
    else return false;
}
◇
```

Macro referenced in 16b.

⟨ implement a callback test for configuration of a digital output 17a ⟩ ≡

```
bool testTwo() {
    description = "configure a digital input";
    pin_settings[6] == s_unknown;
    char *topic = strdup("MyMega/config/dig/6");
    callback(topic, (byte*)"OUTxx", 3);
    free(topic);
    if (pin_settings[6] == s_out) return true;
    else return false;
}
◇
```

Macro referenced in 16b.

⟨ prepare test case 17b ⟩ ≡

```
    TestCallback testCallback1(1);
    Test::add(&testCallback1);
    TestCallback testCallback2(2);
    Test::add(&testCallback2);
◇
```

Macro defined by 8d, 17b, 28a.
Macro referenced in 10b.

⟨ declare dummy version of necessary Arduino library symbols 17c ⟩ ≡

```
#include <EEPROM.h>
#include <Ethernet.h>
EEPROMInterface EEPROM;
EthernetClient Ethernet;
◇
```

Macro defined by 8b, 17c, 29cd, 30ab.
Macro referenced in 10a.

1.7.2 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 17d ⟩ ≡

```
#ifdef USEMQTT
    if (publish_time >= now) {
        for (byte i = 0; i<64; ++i) {
            if (pin_settings[i] == s_in) {
                snprintf(message_buf, 99, "%s/dig/%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 9a.

1.8 The command 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 18a ⟩ ≡

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

Macro defined by 6a, 13e, 18a.
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 18b ⟩ ≡

```
    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 9c, 12ab, 13c, 18b, 28b.
Macro referenced in 5b.

⟨ check and handle command input, return if necessary 19 ⟩ ≡

```
            bool response_required = false;
            const char *error_message = 0;
            int chars_ready = Serial.available();
            if (input_state != command_loaded && chars_ready) {
                ⟨ process serial input 20 ⟩
            }
            else if (input_state == command_loaded) {
    #ifdef DEBUG_CONSOLE
                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 21a ⟩ }
                else if (cmd == 'd') { ⟨ process display command 23d ⟩ }
                else if (cmd == 'h') { ⟨ process host command 21b ⟩ }
                else if (cmd == 'b') { ⟨ process broker command 22a ⟩ }
                else if (cmd == 'p') { ⟨ process port command 23b ⟩ }
                else if (cmd == 's') { ⟨ process save command 23c ⟩ }
                else if (cmd == 'm') { ⟨ process mac address command 22c ⟩ }
                else if (cmd == 'i') { ⟨ process ip address command 23a ⟩ }
                else if (cmd == 'F') { ⟨ process analogue input command 24b ⟩ }
                else if (cmd == 'I') {  ⟨ process digital input command 24a ⟩ }
                else if (cmd == 'O') {  ⟨ process digital output command 25a ⟩ }
            done_command:
            Serial.println(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 9a.

⟨ process serial input 20 ⟩ ≡

```
        int ch = Serial.read();
    #ifdef DEBUG_CONSOLE
        Serial.println(ch);
    #endif
        switch (input_state) {
            case idle:
                if (ch == START_MARK) {
                    input_state = reading;
    #ifdef DEBUG_CONSOLE
                    Serial.print("reading (");
                    Serial.print(chars_ready);
                    Serial.println(")");
    #endif
                }
                break;
            case reading:
                if (ch == END_MARK) {
    #ifdef DEBUG_CONSOLE
                Serial.println("end mark");
    #endif
                    if (input_pos == 0) {
                        input_state = idle; // no command read
    #ifdef DEBUG_CONSOLE
                        Serial.println("idle");
    #endif
                    }
                    else {
                        input_state = command_loaded;
    #ifdef DEBUG_CONSOLE
                        Serial.println("loaded");
    #endif
                    }
    #ifdef DEBUG_CONSOLE
                    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 19.

1.8.1 Command handlers

⟨ process enquiry command 21a ⟩ ≡

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

Macro referenced in 19.

⟨ process host command 21b ⟩ ≡

```
        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 19.

⟨ process broker command 22a ⟩ ≡

```
        scan = 1;
        paramLen = getString(command, scan);
        if (paramLen < 40) {
            strcpy(program_settings.broker_host, paramString);
            Serial.print("broker host set to ");
            Serial.println(paramString);
            DNSClient dns;
            dns.begin(program_settings.dns_address);
            if (!mq_inet_aton(paramString, program_settings.broker_address))
                if (dns.getHostByName(paramString, program_settings.broker_address) != 1)
                    Serial.println("failed to translate broker host to an address");
        }
```
◇

Macro referenced in 19.

⟨ process dns command 22b ⟩ ≡

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

            if (!mq_inet_aton(paramString, program_settings.dns_address)) {
                Serial.println("Failed to initialise dns address");
            }
        }
```
◇

Macro never referenced.

⟨ process mac address command 22c ⟩ ≡

```
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 19.

⟨ process ip address command 23a ⟩ ≡

```
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 19.

⟨ process port command 23b ⟩ ≡

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

Macro referenced in 19.

⟨ process save command 23c ⟩ ≡

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

Macro referenced in 19.

⟨ process display command 23d ⟩ ≡

```
        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 19.

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 24a ⟩ ≡

```
    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 19.

⟨ process analogue input command 24b ⟩ ≡

```
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 19.

⟨ process digital output command 25a ⟩ ≡

```
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 19.

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 25b ⟩ ≡

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

Macro defined by 6c, 13b, 25b, 26ac, 28c, 29a.
Macro referenced in 5b.

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

⟨ function implementations 25c ⟩ ≡

```
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 7ade, 8ac, 11ab, 14a, 16b, 25c, 26b, 27ab, 28d, 29b.
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 26a ⟩ ≡

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

Macro defined by 6c, 13b, 25b, 26ac, 28c, 29a.
Macro referenced in 5b.

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

⟨ function implementations 26b ⟩ ≡

```
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_CONSOLE
        Serial.print("hex: ");
        Serial.print(res);
        Serial.print(" ");
#endif
        ++offset;
        p++;
        ch = upper(*p);
    }
    return res;
}
◇
```

Macro defined by 7ade, 8ac, 11ab, 14a, 16b, 25c, 26b, 27ab, 28d, 29b.
Macro referenced in 5b.

⟨function declarations 26c⟩ ≡

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

Macro defined by 6c, 13b, 25b, 26ac, 28c, 29a.
Macro referenced in 5b.

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

⟨function implementations 27a⟩ ≡

```
    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 7ade, 8ac, 11ab, 14a, 16b, 25c, 26b, 27ab, 28d, 29b.
Macro referenced in 5b.

Test cases

⟨ function implementations 27b ⟩ ≡

```
#ifdef TESTING
class TestGetFloat : public Test {
    int testNum;
    public:
        TestGetFloat(short test) : Test("Test getFloat function",""), 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 7ade, 8ac, 11ab, 14a, 16b, 25c, 26b, 27ab, 28d, 29b.
Macro referenced in 5b.

⟨ prepare test case 28a ⟩ ≡

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

Macro defined by 8d, 17b, 28a.
Macro referenced in 10b.

1.8.3 Reading a string from the PC

⟨ global variables 28b ⟩ ≡

```
char paramString[40];
◇
```

Macro defined by 9c, 12ab, 13c, 18b, 28b.
Macro referenced in 5b.

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

⟨ function declarations 28c ⟩ ≡

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

Macro defined by 6c, 13b, 25b, 26ac, 28c, 29a.
Macro referenced in 5b.

⟨function implementations 28d⟩ ≡

```
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 7ade, 8ac, 11ab, 14a, 16b, 25c, 26b, 27ab, 28d, 29b.
Macro referenced in 5b.

1.9 Utility functions

⟨function declarations 29a⟩ ≡

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

Macro defined by 6c, 13b, 25b, 26ac, 28c, 29a.
Macro referenced in 5b.

⟨function implementations 29b⟩ ≡

```
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 7ade, 8ac, 11ab, 14a, 16b, 25c, 26b, 27ab, 28d, 29b.
Macro referenced in 5b.

1.10 Test Functions

⟨declare dummy version of necessary Arduino library symbols 29c⟩ ≡

```
#define INPUT 0
#define OUTPUT 1
#define LOW 0
#define HIGH 1
#define HEX 0
#define DEC 1
◇
```

Macro defined by 8b, 17c, 29cd, 30ab.
Macro referenced in 10a.

⟨ declare dummy version of necessary Arduino library symbols 29d ⟩ ≡

```
struct SimulatedSerialPort {
    void print(int);
    void println(int);
    void print(float, int);
    void println(float, int);
    void print(const char *);
    void println(const char *);
    void print(const std::string &s);
    void println(const std::string &s);
};
◇
```

Macro defined by 8b, 17c, 29cd, 30ab.
Macro referenced in 10a.

⟨ declare dummy version of necessary Arduino library symbols 30a ⟩ ≡

```
#include <sstream>
struct String {
    std::string s;
    String(const char *str) { s = str; }
    String(unsigned int a, int b) {
        std::stringstream ss;
        ss << a << " " << b;
        s = ss.str();
    }
};
const char *operator+(const char *a, String b) {
    std::string s(a);
    s += b.s;
}
◇
```

Macro defined by 8b, 17c, 29cd, 30ab.
Macro referenced in 10a.

⟨ declare dummy version of necessary Arduino library symbols 30b ⟩ ≡

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

Macro defined by 8b, 17c, 29cd, 30ab.
Macro referenced in 10a.

⟨ implement dummy version of necessary Arduino library symbols 30c ⟩ ≡

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

Macro defined by 30cd.
Macro referenced in 10a.

⟨ implement dummy version of necessary Arduino library symbols 30d ⟩ ≡

```
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"; }
void SimulatedSerialPort::print(const std::string &s) {  std::cout << s; }
void SimulatedSerialPort::println(const std::string &s) {  std::cout << s << "\n"; }
```

◇

Macro defined by 30cd.
Macro referenced in 10a.

Chapter 2

Installation

2.1 Generating the program from the source file

⟨ compile the document using nuweb 31 ⟩ ≡

```
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 10a.
`"mquino.cpp"` Defined by 5a.
`"test_driver.cpp"` Defined by 10b.

Appendix B

Macros

⟨ check and handle command input, return if necessary 19 ⟩ Referenced in 9a.
⟨ check inputs for change of state or publish timer and publish their status 17d ⟩ Referenced in 9a.
⟨ check the connection and connect if necessary 12d ⟩ Referenced in 9a.
⟨ classes and structures 10c ⟩ Referenced in 5b.
⟨ compile the document using nuweb 31 ⟩ Not referenced.
⟨ constants and type definitions 6a, 13e, 18a ⟩ Referenced in 5b.
⟨ declarations and functions 5b ⟩ Referenced in 5a, 10b.
⟨ declare dummy version of necessary Arduino library symbols 8b, 17c, 29cd, 30ab ⟩ Referenced in 10a.
⟨ declare global variables that must be declared first 7b ⟩ Referenced in 5b.
⟨ declare local shared variables ? ⟩ Referenced in 9a.
⟨ function declarations 6c, 13b, 25b, 26ac, 28c, 29a ⟩ Referenced in 5b.
⟨ function implementations 7ade, 8ac, 11ab, 14a, 16b, 25c, 26b, 27ab, 28d, 29b ⟩ Referenced in 5b.
⟨ get the current time into variable 'now' 9d ⟩ Referenced in 9a.
⟨ global variables 9c, 12ab, 13c, 18b, 28b ⟩ Referenced in 5b.
⟨ implement a callback test for configuration of a digital input 16c ⟩ Referenced in 16b.
⟨ implement a callback test for configuration of a digital output 17a ⟩ Referenced in 16b.
⟨ implement dummy version of necessary Arduino library symbols 30cd ⟩ Referenced in 10a.
⟨ include other headers and conditional code macros 5d, 6b, 11cd ⟩ Referenced in 5b.
⟨ poll MQTT 13a ⟩ Referenced in 9a.
⟨ prepare test case 8d, 17b, 28a ⟩ Referenced in 10b.
⟨ process analogue input command 24b ⟩ Referenced in 19.
⟨ process broker command 22a ⟩ Referenced in 19.
⟨ process digital input command 24a ⟩ Referenced in 19.
⟨ process digital output command 25a ⟩ Referenced in 19.
⟨ process display command 23d ⟩ Referenced in 19.
⟨ process dns command 22b ⟩ Not referenced.
⟨ process enquiry command 21a ⟩ Referenced in 19.
⟨ process host command 21b ⟩ Referenced in 19.
⟨ process ip address command 23a ⟩ Referenced in 19.
⟨ process mac address command 22c ⟩ Referenced in 19.
⟨ process port command 23b ⟩ Referenced in 19.
⟨ process save command 23c ⟩ Referenced in 19.
⟨ process serial input 20 ⟩ Referenced in 19.
⟨ process the current field 14b ⟩ Referenced in 14a.
⟨ program initialisation steps 7c, 9e, 12c, 13d ⟩ Referenced in 5c.
⟨ protect against clock wrap-around ? ⟩ Referenced in 9a.
⟨ publish changes on an input pin 16a ⟩ Referenced in 14b.
⟨ shared class and structure definitions ? ⟩ Referenced in 5b.
⟨ special microcontroller initialisation 9b ⟩ Referenced in 5c.
⟨ subscribe to the topic that indicates changes on an output pin 15 ⟩ Referenced in 14b.
⟨ the main loop function 9a ⟩ Referenced in 5a.
⟨ the setup function 5c ⟩ Referenced in 5a.

Appendix C

Identifiers

Bibliography