Arduino MQTT Interface

Martin Leadbeater

March 26, 2013

Contents

1 Introduction **5**
- 1.1 Command processing . . . 6
- 1.2 Program settings . . . 6
  - 1.2.1 Test cases . . . 9
- 1.3 The main loop . . . 10
- 1.4 Loop timer . . . 10
- 1.5 Testing . . . 11
- 1.6 Debug Reporting . . . 12
- 1.7 MQTT Interface . . . 13
  - 1.7.1 Test cases . . . 19
  - 1.7.2 Publishing updates . . . 20
- 1.8 The command parser . . . 21
  - 1.8.1 Command handlers . . . 23
  - 1.8.2 Reading a number from the PC . . . 27
  - 1.8.3 Reading a string from the PC . . . 30
- 1.9 Utility functions . . . 31
- 1.10 Test Functions . . . 31

2 Installation **35**
- 2.1 Generating the program from the source file . . . 35

A Files **37**

B Macros **39**

C Identifiers **41**

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 10b ⟩
◇
```

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 6, ... ⟩
⟨ constants and type definitions 7a, ... ⟩
⟨ shared class and structure definitions ? ⟩
⟨ classes and structures 12a ⟩
⟨ function declarations 7c, ... ⟩
⟨ declare global variables that must be declared first 8b ⟩
⟨ global variables 10d, ... ⟩
⟨ function implementations 8a, ... ⟩
◇
```

Macro referenced in 5a, 11b.

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 10c ⟩
    ⟨ program initialisation steps 10f, ... ⟩
}
◇
```

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 6 ⟩ ≡

```
#include <EEPROM.h>
```

◇

Macro defined by 6, 7b, 13ab.
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 7a ⟩ ≡

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

◇

Macro defined by 7a, 15c, 21b.
Macro referenced in 5b.

The initialisation requires a host lookup

⟨ include other headers and conditional code macros 7b ⟩ ≡

```
#include <Dns.h>
```

◇

Macro defined by 6, 7b, 13ab.
Macro referenced in 5b.

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

⟨ function declarations 7c ⟩ ≡

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

◇

Macro defined by 7c, 14d, 18d, 27c, 28b, 29a, 30d, 31b.
Macro referenced in 5b.

Note that we assume the broker host is null terminated. The `init` method takes care of initialising the EEPROM if necessary and it updates the broker ip address from the name but doesn't bother to detect a difference from last time and save that ip if it changed.

⟨ function implementations 8a ⟩ ≡

```
void ProgramSettings::init(EthernetClient &enet_client) {
    bool need_save = false;
    load();
    if (!program_settings.valid()) {
        need_save = true;
        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;
    }
    Ethernet.begin(mac_address);
    DNSClient dns;
    dns.begin(dns_address);
    if (dns.getHostByName(broker_host, broker_address) == 1) {
        for (int i=0; i<4; ++i) broker_ip[i] = broker_address[i];
    }
    if (need_save)
        program_settings.save();
}
◇
```

Macro defined by 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac.
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 8b ⟩ ≡

```
EthernetClient enet_client;
ProgramSettings program_settings;
◇
```

Macro referenced in 5b.

⟨ load program settings 8c ⟩ ≡

```
    program_settings.init(enet_client);
◇
```

Macro referenced in 14a.

⟨ function implementations 8d ⟩ ≡

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

Macro defined by 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac.
Macro referenced in 5b.

⟨ function implementations 9a ⟩ ≡

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

Macro defined by 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac.
Macro referenced in 5b.

⟨ function implementations 9b ⟩ ≡

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

Macro defined by 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac.
Macro referenced in 5b.

1.2.1 Test cases

⟨ declare dummy version of necessary Arduino library symbols 9c ⟩ ≡

```
typedef uint8_t byte;
```
◇

Macro defined by 9c, 20d, 31d, 32abc.
Macro referenced in 11a.

⟨ function implementations 9d ⟩ ≡

```
#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 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac.
Macro referenced in 5b.

⟨ prepare test case 10a ⟩ ≡

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

Macro defined by 10a, 20c, 30b.
Macro referenced in 11b.

1.3 The main loop

⟨ the main loop function 10b ⟩ ≡

```
void loop() {
    ⟨ declare local shared variables ? ⟩
    ⟨ check the connection and connect if necessary 14b ⟩
    ⟨ poll MQTT 14c ⟩
    ⟨ get the current time into variable 'now' 10e ⟩
    ⟨ protect against clock wrap-around ? ⟩
    ⟨ check and handle command input, return if necessary 22 ⟩
    ⟨ check inputs for change of state or publish timer and publish their status 21a ⟩
}
```
◇

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 10c ⟩ ≡

```
    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 10d ⟩ ≡

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

Macro defined by 10d, 13cd, 15a, 21c, 30c.
Macro referenced in 5b.

⟨ get the current time into variable 'now' 10e ⟩ ≡

```
now = millis();
```
◇

Macro referenced in 10b.

⟨ program initialisation steps 10f ⟩ ≡

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

Macro defined by 10f, 14a, 15b.
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" 11a ≡

```
#include <iostream>
#define TESTING 1
⟨ declare dummy version of necessary Arduino library symbols 9c, ... ⟩
⟨ implement dummy version of necessary Arduino library symbols 32d, ... ⟩
◇
```

"test_driver.cpp" 11b ≡

```
#include "arduino_stubs.h"
#include <iostream>
#include <list>
⟨ declarations and functions 5b ⟩
int main(int argc, char *argv[]) {
    ⟨ prepare test case 10a, ... ⟩
    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 12a ⟩ ≡

```
#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 12b ⟩ ≡

```
#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 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac.
Macro referenced in 5b.

⟨ function implementations 12c ⟩ ≡

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

Macro defined by 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac.
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 13a ⟩ ≡

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

Macro defined by 6, 7b, 13ab.
Macro referenced in 5b.

1.7 MQTT Interface

⟨ include other headers and conditional code macros 13b ⟩ ≡

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

Macro defined by 6, 7b, 13ab.
Macro referenced in 5b.

⟨ global variables 13c ⟩ ≡

```
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 10d, 13cd, 15a, 21c, 30c.
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 13d ⟩ ≡

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

PubSubClient client(enet_client);
```
◇

Macro defined by 10d, 13cd, 15a, 21c, 30c.
Macro referenced in 5b.

⟨ initialise MQTT client 13e ⟩ ≡

```
    client.setCallback(callback);
    client.setServerIP(program_settings.broker_ip);
    client.setPort(program_settings.broker_port);
```
◇

Macro referenced in 14a.

Initialise the ethernet MAC address and MQTT client.

⟨ program initialisation steps 14a ⟩ ≡

```
#ifdef USEMQTT
  ⟨ load program settings 8c ⟩
  ⟨ initialise MQTT client 13e ⟩
  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 10f, 14a, 15b.
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 14b ⟩ ≡

```
#ifdef USEMQTT
  if (!client.connected())
  {
      // clientID, username, MD5 encoded password
      client.connect("mquino", "mquino_user", "00000000000000000000000000000000");
      if (!client.connected()) Serial.println("connection failed");
      else {
          snprintf(config_topic, 29, "%s/config/dig/+", program_settings.hostname);
          client.subscribe(config_topic);
          snprintf(config_topic, 29, "%s/config/ana/+", program_settings.hostname);
          client.subscribe(config_topic);
      }
  }
#endif
```

◇

Macro referenced in 10b.

⟨ poll MQTT 14c ⟩ ≡

```
    if (client.connected()) client.loop();
```

◇

Macro referenced in 10b.

Subscribed data arrives via a callback

⟨ function declarations 14d ⟩ ≡

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

◇

Macro defined by 7c, 14d, 18d, 27c, 28b, 29a, 30d, 31b.
Macro referenced in 5b.

The program expects messages in one of three 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.

topic	*message*
name '/' ``config'' '/' ``dig'' '/'	``IN'' or ``OUT'' or
pin_number	``PWM'' or ``OFF''
name '/' ``config'' '/' ``ana'' '/'	``AIN'' or ``OFF''
pin_number	
name '/' ``dig'' '/' ``pin_number''	``on'' or ``off'' or value

Table 1.2: Expected message formats

⟨ global variables 15a ⟩ ≡

```
#ifdef USEMQTT
int pin_settings[72];
#endif
```

◇

Macro defined by 10d, 13cd, 15a, 21c, 30c.
Macro referenced in 5b.

⟨ program initialisation steps 15b ⟩ ≡

```
#ifdef USEMQTT
    for(int i=0; i<72; ++i) pin_settings[i] = s_unknown;
#endif
```

◇

Macro defined by 10f, 14a, 15b.
Macro referenced in 5c.

⟨ constants and type definitions 15c ⟩ ≡

```
#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 7a, 15c, 21b.
Macro referenced in 5b.

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

⟨ function implementations 16 ⟩ ≡

```
#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);
  bool analogue_pin = false; // changes to true if the topic refers to an analogue pin
  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 17 ⟩

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

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

◇

Macro defined by 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac.
Macro referenced in 5b.

⟨ process the current field 17 ⟩ ≡

```
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 if (strcmp(message_buf, "ana") == 0 && parse_state == ps_processing_config) {
        analogue_pin = true;
        field = f_pin; // found analogue config
    }
    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, "IGNORE", length) == 0) setting = s_unknown;
    else if (strncmp((const char *)payload, "AIN", length) == 0) setting = s_value;
    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 18a ⟩
        }
        else if (setting == s_in) {
            ⟨ publish changes on an input pin 18b ⟩
        }
        else if (setting == s_value) {
            ⟨ publish changes on an analogue input pin 19b ⟩
        }
        else if (setting == s_unknown) {
            ⟨ stop publishing changes on the input 18c ⟩
        }
        else if (setting == s_pwm) {
            Serial.println ("PWM mode is not currently supported");
        }
    }
    else if (parse_state == ps_setting_output) {
```

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 18a ⟩ ≡

```
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 17.

⟨ publish changes on an input pin 18b ⟩ ≡

```
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 17.

⟨ stop publishing changes on the input 18c ⟩ ≡

```
if (analogue_pin) {
    if (pin < 8) pin_settings[64 + pin] = s_unknown;
    Serial.print("stopped publishing analogue pin ");
}
else if (pin < 64) {
    Serial.print("stopped publishing pin ");
    pin_settings[pin] = s_unknown;
}
Serial.println(pin);
```

◇

Macro referenced in 17.

⟨ function declarations 18d ⟩ ≡

```
int readAnalogueValue(int pin);
```

◇

Macro defined by 7c, 14d, 18d, 27c, 28b, 29a, 30d, 31b.
Macro referenced in 5b.

⟨ function implementations 19a ⟩ ≡

```
int readAnalogueValue(int pin)
{
    int value = 0;
    if (pin == 0) value = analogRead(A0);
    else if (pin == 1) value = analogRead(A1);
    else if (pin == 2) value = analogRead(A2);
    else if (pin == 3) value = analogRead(A3);
    else if (pin == 4) value = analogRead(A4);
    else if (pin == 5) value = analogRead(A5);
    else if (pin == 6) value = analogRead(A6);
    else if (pin == 7) value = analogRead(A7);
    return value;
}
```

◇

Macro defined by 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac.
Macro referenced in 5b.

⟨ publish changes on an analogue input pin 19b ⟩ ≡

```
if (pin < 8) {
    pin_settings[64 + pin] = s_value;
    snprintf(message_buf, 99, "%s/ana/%d", program_settings.hostname, pin);
    char status[10];
    int value = readAnalogueValue(pin);
    snprintf(status, 10, "%d", value);
#ifdef FEEDBACK
    Serial.print("publishing ");
    Serial.print(value);
    Serial.print(" to: ");
    Serial.println(message_buf);
#endif
    client.publish(message_buf, (uint8_t*)status, strlen(status), true );
}
```

◇

Macro referenced in 17.

1.7.1 Test cases

⟨ function implementations 19c ⟩ ≡

```
#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 20a ⟩
        ⟨ implement a callback test for configuration of a digital output 20b ⟩
};
#endif
```

◇

Macro defined by 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac.
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 20a ⟩ ≡

```
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 19c.

⟨ implement a callback test for configuration of a digital output 20b ⟩ ≡

```
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 19c.

⟨ prepare test case 20c ⟩ ≡

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

◇

Macro defined by 10a, 20c, 30b.
Macro referenced in 11b.

⟨ declare dummy version of necessary Arduino library symbols 20d ⟩ ≡

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

◇

Macro defined by 9c, 20d, 31d, 32abc.
Macro referenced in 11a.

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 21a ⟩ ≡

```
#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 );
            Serial.print(message_buf);
            Serial.print("\t");
            Serial.println(status);
            }
        }
        for (byte i = 0; i<8; ++i) {
            if (pin_settings[64 + i] == s_value) {
                snprintf(message_buf, 99, "%s/ana/%d", program_settings.hostname, i);
                char status[10];
                int value = readAnalogueValue(i);
                snprintf(status, 10, "%d", value);
                client.publish(message_buf, (uint8_t*)status, strlen(status), true );
            Serial.print(message_buf);
            Serial.print("\t");
            Serial.println(status);
            }
        }
        publish_time += 1000;
    }
#endif
◇
```

Macro referenced in 10b.

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 21b ⟩ ≡

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

Macro defined by 7a, 15c, 21b.
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 21c ⟩ ≡

```
    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 10d, 13cd, 15a, 21c, 30c.
Macro referenced in 5b.

⟨ check and handle command input, return if necessary 22 ⟩ ≡

```
        bool response_required = false;
        const char *error_message = 0;
        int chars_ready = Serial.available();
        if (input_state != command_loaded && chars_ready) {
            ⟨ process serial input 23a ⟩
        }
        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 23b ⟩ }
            else if (cmd == 'd') { ⟨ process display command 26a ⟩ }
            else if (cmd == 'h') { ⟨ process host command 24a ⟩ }
            else if (cmd == 'b') { ⟨ process broker command 24b ⟩ }
            else if (cmd == 'p') { ⟨ process port command 25c ⟩ }
            else if (cmd == 's') { ⟨ process save command 25d ⟩ }
            else if (cmd == 'm') { ⟨ process mac address command 25a ⟩ }
            else if (cmd == 'i') { ⟨ process ip address command 25b ⟩ }
            else if (cmd == 'F') { ⟨ process analogue input command 27a ⟩ }
            else if (cmd == 'I') { ⟨ process digital input command 26b ⟩ }
            else if (cmd == 'O') { ⟨ process digital output command 27b ⟩ }
        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 10b.

⟨ process serial input 23a ⟩ ≡

```
    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 22.

1.8.1 Command handlers

⟨ process enquiry command 23b ⟩ ≡

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

◇

Macro referenced in 22.

⟨ process host command 24a ⟩ ≡

```
        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 22.

⟨ process broker command 24b ⟩ ≡

```
        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 22.

⟨ process dns command 24c ⟩ ≡

```
        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 25a ⟩ ≡

```
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 22.

⟨ process ip address command 25b ⟩ ≡

```
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 22.

⟨ process port command 25c ⟩ ≡

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

◇

Macro referenced in 22.

⟨ process save command 25d ⟩ ≡

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

◇

Macro referenced in 22.

⟨ process display command 26a ⟩ ≡

```
    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 22.

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 26b ⟩ ≡

```
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 22.

⟨ process analogue input command 27a ⟩ ≡

```
if (param1 >= 0 && param1 <= 7) {
    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 if (param1 == 6) param1 = A6;
    else if (param1 == 7) param1 = A7;
    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..7";
```

◇

Macro referenced in 22.

⟨ process digital output command 27b ⟩ ≡

```
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 22.

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 27c ⟩ ≡

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

◇

Macro defined by 7c, 14d, 18d, 27c, 28b, 29a, 30d, 31b.
Macro referenced in 5b.

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

⟨ function implementations 28a ⟩ ≡

```
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 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac.
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 28b ⟩ ≡

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

◇

Macro defined by 7c, 14d, 18d, 27c, 28b, 29a, 30d, 31b.
Macro referenced in 5b.

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

⟨ function implementations 28c ⟩ ≡

```
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 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac.
Macro referenced in 5b.

⟨ function declarations 29a ⟩ ≡

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

◇

Macro defined by 7c, 14d, 18d, 27c, 28b, 29a, 30d, 31b.
Macro referenced in 5b.

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

⟨ function implementations 29b ⟩ ≡

```
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 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac.
Macro referenced in 5b.

Test cases

⟨ function implementations 30a ⟩ ≡

```
#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 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac.
Macro referenced in 5b.

⟨ prepare test case 30b ⟩ ≡

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

◇

Macro defined by 10a, 20c, 30b.
Macro referenced in 11b.

1.8.3 Reading a string from the PC

⟨ global variables 30c ⟩ ≡

```
char paramString[40];
```

◇

Macro defined by 10d, 13cd, 15a, 21c, 30c.
Macro referenced in 5b.

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

⟨ function declarations 30d ⟩ ≡

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

◇

Macro defined by 7c, 14d, 18d, 27c, 28b, 29a, 30d, 31b.
Macro referenced in 5b.

⟨ function implementations 31a ⟩ ≡

```
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 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac.
Macro referenced in 5b.

1.9 Utility functions

⟨ function declarations 31b ⟩ ≡

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

◇

Macro defined by 7c, 14d, 18d, 27c, 28b, 29a, 30d, 31b.
Macro referenced in 5b.

⟨ function implementations 31c ⟩ ≡

```
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 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac.
Macro referenced in 5b.

1.10 Test Functions

⟨ declare dummy version of necessary Arduino library symbols 31d ⟩ ≡

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

◇

Macro defined by 9c, 20d, 31d, 32abc.
Macro referenced in 11a.

⟨ declare dummy version of necessary Arduino library symbols 32a ⟩ ≡

```
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 9c, 20d, 31d, 32abc.
Macro referenced in 11a.

⟨ declare dummy version of necessary Arduino library symbols 32b ⟩ ≡

```
#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 9c, 20d, 31d, 32abc.
Macro referenced in 11a.

⟨ declare dummy version of necessary Arduino library symbols 32c ⟩ ≡

```
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 9c, 20d, 31d, 32abc.
Macro referenced in 11a.

⟨ implement dummy version of necessary Arduino library symbols 32d ⟩ ≡

```
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 32d, 33.
Macro referenced in 11a.

⟨ implement dummy version of necessary Arduino library symbols 33 ⟩ ≡

```
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 32d, 33.
Macro referenced in 11a.

Chapter 2

Installation

2.1 Generating the program from the source file

⟨ compile the document using nuweb 35 ⟩ ≡

```
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 11a.
"mquino.cpp" Defined by 5a.
"test_driver.cpp" Defined by 11b.

Appendix B

Macros

⟨ check and handle command input, return if necessary 22 ⟩ Referenced in 10b.
⟨ check inputs for change of state or publish timer and publish their status 21a ⟩ Referenced in 10b.
⟨ check the connection and connect if necessary 14b ⟩ Referenced in 10b.
⟨ classes and structures 12a ⟩ Referenced in 5b.
⟨ compile the document using nuweb 35 ⟩ Not referenced.
⟨ constants and type definitions 7a, 15c, 21b ⟩ Referenced in 5b.
⟨ declarations and functions 5b ⟩ Referenced in 5a, 11b.
⟨ declare dummy version of necessary Arduino library symbols 9c, 20d, 31d, 32abc ⟩ Referenced in 11a.
⟨ declare global variables that must be declared first 8b ⟩ Referenced in 5b.
⟨ declare local shared variables ? ⟩ Referenced in 10b.
⟨ function declarations 7c, 14d, 18d, 27c, 28b, 29a, 30d, 31b ⟩ Referenced in 5b.
⟨ function implementations 8ad, 9abd, 12bc, 16, 19ac, 28ac, 29b, 30a, 31ac ⟩ Referenced in 5b.
⟨ get the current time into variable 'now' 10e ⟩ Referenced in 10b.
⟨ global variables 10d, 13cd, 15a, 21c, 30c ⟩ Referenced in 5b.
⟨ implement a callback test for configuration of a digital input 20a ⟩ Referenced in 19c.
⟨ implement a callback test for configuration of a digital output 20b ⟩ Referenced in 19c.
⟨ implement dummy version of necessary Arduino library symbols 32d, 33 ⟩ Referenced in 11a.
⟨ include other headers and conditional code macros 6, 7b, 13ab ⟩ Referenced in 5b.
⟨ initialise MQTT client 13e ⟩ Referenced in 14a.
⟨ load program settings 8c ⟩ Referenced in 14a.
⟨ poll MQTT 14c ⟩ Referenced in 10b.
⟨ prepare test case 10a, 20c, 30b ⟩ Referenced in 11b.
⟨ process analogue input command 27a ⟩ Referenced in 22.
⟨ process broker command 24b ⟩ Referenced in 22.
⟨ process digital input command 26b ⟩ Referenced in 22.
⟨ process digital output command 27b ⟩ Referenced in 22.
⟨ process display command 26a ⟩ Referenced in 22.
⟨ process dns command 24c ⟩ Not referenced.
⟨ process enquiry command 23b ⟩ Referenced in 22.
⟨ process host command 24a ⟩ Referenced in 22.
⟨ process ip address command 25b ⟩ Referenced in 22.
⟨ process mac address command 25a ⟩ Referenced in 22.
⟨ process port command 25c ⟩ Referenced in 22.
⟨ process save command 25d ⟩ Referenced in 22.
⟨ process serial input 23a ⟩ Referenced in 22.
⟨ process the current field 17 ⟩ Referenced in 16.
⟨ program initialisation steps 10f, 14a, 15b ⟩ Referenced in 5c.
⟨ protect against clock wrap-around ? ⟩ Referenced in 10b.
⟨ publish changes on an analogue input pin 19b ⟩ Referenced in 17.
⟨ publish changes on an input pin 18b ⟩ Referenced in 17.
⟨ shared class and structure definitions ? ⟩ Referenced in 5b.
⟨ special microcontroller initialisation 10c ⟩ Referenced in 5c.
⟨ stop publishing changes on the input 18c ⟩ Referenced in 17.
⟨ subscribe to the topic that indicates changes on an output pin 18a ⟩ Referenced in 17.
⟨ the main loop function 10b ⟩ Referenced in 5a.
⟨ the setup function 5c ⟩ Referenced in 5a.

Appendix C

Identifiers

`loop`: 5a, 10b, 14b, 14c.
`setup`: 5a, 5c, 8a.

Bibliography