mcu-uuid-telnet

Simon Arlott

Nov 26, 2022

CONTENTS

CHAPTER

ONE

DESCRIPTION

Microcontroller telnet service

CHAPTER

TWO

PURPOSE

Provides access to a console shell as a telnet server (using the RFC 854 protocol).

CHAPTER

THREE

DEPENDENCIES

- mcu-uuid-console

Refer to the `library.json` file for more details.

CHAPTER

FOUR

CONTENTS

4.1 Usage

```
#include <uuid/telnet.h>
```

Create a `uuid::telnet::TelnetService` and call `start()`.

Call `loop()` regularly and when WiFi connectivity is available connections will be accepted.

Call `uuid::console::Shell::loop_all()` regularly to process open connections.

The binary, echo and suppress go ahead telnet options will be negotiated.

4.1.1 Example

```
#include <Arduino.h>
#ifdef ARDUINO_ARCH_ESP8266
# include <ESP8266WiFi.h>
#else
# include <WiFi.h>
#endif

#include <memory>
#include <string>
#include <vector>

#include <uuid/common.h>
#include <uuid/console.h>
#include <uuid/telnet.h>

using uuid::read_flash_string;
using uuid::flash_string_vector;
using uuid::console::Commands;
using uuid::console::Shell;

static std::shared_ptr<Commands> commands = std::make_shared<Commands>();
static uuid::telnet::TelnetService telnet{commands};

void setup() {
        commands->add_command(flash_string_vector{F("pinMode")},
```

(continues on next page)

(continued from previous page)

```
                flash_string_vector{F("<pin>"), F("<mode>")},

                [] (Shell &shell, const std::vector<std::string> &arguments) {
                        uint8_t pin = String(arguments[0].c_str()).toInt();
                        uint8_t mode;

                        if (arguments[1] == read_flash_string(F("INPUT"))) {
                                mode = INPUT;
                        } else if (arguments[1] == read_flash_string(F("OUTPUT"))) {
                                mode = OUTPUT;
                        } else if (arguments[1] == read_flash_string(F("INPUT_PULLUP")))
↪{
                                mode = INPUT_PULLUP;
                        } else {
                                shell.println(F("Invalid mode"));
                                return;
                        }

                        pinMode(pin, mode);
                        shell.printfln(F("Configured pin %u to mode %s"),
                                        pin, arguments[1].c_str());
                },

                [] (Shell &shell, const std::vector<std::string> &current_arguments,
                                const std::string &next_argument)
                                -> const std::vector<std::string> {
                        if (current_arguments.size() == 1) {
                                /* The first argument has been provided, so return
                                 * completion values for the second argument.
                                 */
                                return {
                                        read_flash_string(F("INPUT")),
                                        read_flash_string(F("OUTPUT")),
                                        read_flash_string(F("INPUT_PULLUP"))
                                };
                        } else {
                                return {};
                        }
                }
        );

        commands->add_command(flash_string_vector{F("digitalRead")},
                flash_string_vector{F("<pin>")},

                [] (Shell &shell, const std::vector<std::string> &arguments) {
                        uint8_t pin = String(arguments[0].c_str()).toInt();
                        auto value = digitalRead(pin);

                        shell.printfln(F("Read value from pin %u: %S"),
                                        pin, value == HIGH ? F("HIGH") : F("LOW"));
                }
        );
```

(continues on next page)

(continued from previous page)

```
commands->add_command(flash_string_vector{F("digitalWrite")},
        flash_string_vector{F("<pin>"), F("<value>")},

        [] (Shell &shell, const std::vector<std::string> &arguments) {
                uint8_t pin = String(arguments[0].c_str()).toInt();
                uint8_t value;

                if (arguments[1] == read_flash_string(F("HIGH"))) {
                        value = HIGH;
                } else if (arguments[1] == read_flash_string(F("LOW"))) {
                        value = LOW;
                } else {
                        shell.println(F("Invalid value"));
                        return;
                }

                digitalWrite(pin, value);
                shell.printfln(F("Wrote %s value to pin %u"),
                                arguments[1].c_str(), pin);
        },

        [] (Shell &shell, const std::vector<std::string> &current_arguments,
                const std::string &next_argument)
                -> const std::vector<std::string> {
                if (current_arguments.size() == 1) {
                        /* The first argument has been provided, so return
                         * completion values for the second argument.
                         */
                        return {
                                read_flash_string(F("HIGH")),
                                read_flash_string(F("LOW"))
                        };
                } else {
                        return {};
                }
        }
);

commands->add_command(flash_string_vector{F("help")},
        [] (Shell &shell, const std::vector<std::string> &arguments) {
                shell.print_all_available_commands();
        }
);

commands->add_command(flash_string_vector{F("exit")},
        [] (Shell &shell, const std::vector<std::string> &arguments) {
                shell.stop();
        }
);

telnet.start();
```

(continues on next page)

(continued from previous page)

```
        WiFi.persistent(false);
        WiFi.mode(WIFI_STA);
        WiFi.begin("SSID", "password");

        Serial.begin(115200);
}

void loop() {
        uuid::loop();
        telnet.loop();
        Shell::loop_all();
        yield();
}
```

CHAPTER

FIVE

RESOURCES

5.1 Change log

5.1.1 Unreleased

5.1.2 0.1.5 – 2022-11-26

Upgrade version of `uuid-console`.

Changed

- Use `PSTR_ALIGN` for flash strings.
- Use version 2.x.x of `uuid-console`.

5.1.3 0.1.4 – 2022-10-29

Upgrade version of `uuid-console`.

Changed

- Use version 1.x.x of `uuid-console`.

5.1.4 0.1.3 – 2022-07-12

Upgrade version of `uuid-console`.

Changed

- Use version 0.9.x of `uuid-console`.

5.1.5 0.1.2 – 2021-04-18

Upgrade to PlatformIO 5.

Changed

- Use PlatformIO 5 dependency specification.

5.1.6 0.1.1 – 2021-01-17

Upgrade to the latest version of the dependencies for static initialization and deinitialization fixes.

5.1.7 0.1.0 – 2019-09-16

Initial development release.

Added

- Telnet listening service.
- Telnet stream handler with option negotiation.
- Use of TCP keepalives to timeout open connections.
- Configurable idle timeout.
- Configurable write timeout.