mcu-uuid-syslog

Simon Arlott

Nov 07, 2022

CONTENTS

CHAPTER

ONE

DESCRIPTION

Microcontroller syslog service

CHAPTER

TWO

PURPOSE

Provides a log handler that sends messages to a syslog server (using the RFC 5424 protocol). Thread-safe (for log messages only, not configuration) on the ESP32.

CHAPTER

THREE

DEPENDENCIES

- mcu-uuid-common
- mcu-uuid-log

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

CHAPTER

FOUR

CONTENTS

4.1 Usage

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

Create a `uuid::syslog::SyslogService` and call `start()`. Then set the destination host, log level and optional local hostname.

The default log level is `uuid::log::Level::ALL` so it can capture messages logged during startup before any configuration is read.

Call `loop()` regularly and when WiFi connectivity is available the queued messages will be sent.

On ESP8266 and ESP32 platforms an ARP query will be made every second until a route to the destination host is available, before sending messages at a rate of 100 per second. On other platforms messages will be sent at a rate of 10 per second to allow time for ARP lookups to complete and avoid dropping messages.

Whenever possible, the current system time is included in the outgoing syslog message.

4.1.1 Example

```
#include <Arduino.h>
#ifdef ARDUINO_ARCH_ESP8266
# include <ESP8266WiFi.h>
#else
# include <WiFi.h>
#endif
#include <uuid/common.h>
#include <uuid/log.h>
#include <uuid/syslog.h>

static uuid::syslog::SyslogService syslog;

void setup() {
        static uuid::log::Logger logger{F("setup")};

        syslog.start();
        /* Retrieve settings after calling start()
         * in case the settings read process logs
         * some messages.
         */
```

(continues on next page)

(continued from previous page)

```
        syslog.hostname("example");
        syslog.log_level(uuid::log::DEBUG);
        syslog.mark_interval(3600);
        syslog.destination(IPAddress(192, 0, 2, 1));

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

        Serial.begin(115200);

        logger.info(F("Application started"));
}

void loop() {
        static uuid::log::Logger logger{F("loop")};
        static unsigned int i = 0;

        uuid::loop();
        syslog.loop();

        logger.debug(F("Hello %u World!"), i++);

        delay(1000);
}
```

Output

```
UDP -> [192.0.2.1]:514 <46>1 - example - - - - 000+00:00:00.000 I 0: [syslog] Log level␣
↪set to DEBUG
UDP -> [192.0.2.1]:514 <134>1 - example - - - - 000+00:00:00.000 I 1: [setup]␣
↪Application started
UDP -> [192.0.2.1]:514 <135>1 - example - - - - 000+00:00:00.000 D 2: [loop] Hello 0␣
↪World!
UDP -> [192.0.2.1]:514 <135>1 - example - - - - 000+00:00:01.000 D 3: [loop] Hello 1␣
↪World!
UDP -> [192.0.2.1]:514 <135>1 - example - - - - 000+00:00:02.000 D 4: [loop] Hello 2␣
↪World!
UDP -> [192.0.2.1]:514 <135>1 2020-01-01T00:00:00.000000Z example - - - - 000+00:00:03.
↪000 D 5: [loop] Hello 3 World!
UDP -> [192.0.2.1]:514 <135>1 2020-01-01T00:00:01.000000Z example - - - - 000+00:00:04.
↪000 D 6: [loop] Hello 4 World!
UDP -> [192.0.2.1]:514 <135>1 2020-01-01T00:00:02.000000Z example - - - - 000+00:00:05.
↪000 D 7: [loop] Hello 5 World!
UDP -> [192.0.2.1]:514 <135>1 2020-01-01T00:00:03.000000Z example - - - - 000+00:00:06.
↪000 D 8: [loop] Hello 6 World!
UDP -> [192.0.2.1]:514 <135>1 2020-01-01T00:00:04.000000Z example - - - - 000+00:00:07.
↪000 D 9: [loop] Hello 7 World!
UDP -> [192.0.2.1]:514 <135>1 2020-01-01T00:00:05.000000Z example - - - - 000+00:00:08.
↪000 D 10: [loop] Hello 8 World!
```

(continues on next page)

(continued from previous page)

```
UDP -> [192.0.2.1]:514 <135>1 2020-01-01T00:00:06.000000Z example - - - - 000+00:00:09.
↪000 D 11: [loop] Hello 9 World!
UDP -> [192.0.2.1]:514 <135>1 2020-01-01T00:00:07.000000Z example - - - - 000+00:00:10.
↪000 D 12: [loop] Hello 10 World!
```

CHAPTER

FIVE

RESOURCES

5.1 Change log

5.1.1 Unreleased

5.1.2 2.2.2 – 2022-11-07

Fix handling of (invalid) facility values higher than 31.

Changed

- Use `PSTR_ALIGN` for flash strings.

Fixed

- Facility values higher than 31 (values higher than 23 are not allowed by the `uuid::log::Facility` enum) would cause the priority value to be out of range (more than 3 digits).

5.1.3 2.2.1 – 2022-10-29

Fix thread-safe mode and mark message handling.

Changed

- Limit the maximum number of log messages that can be transmitted in one loop iteration.

Fixed

- Crash when transmitting messages when operating in thread-safe mode.
- Last message time wasn't being updated so a flood of mark messages would occur when the uptime reached the mark interval.

5.1.4 2.2.0 – 2022-10-26

Be thread-safe (for log messages) where possible.

Added

- Indicate whether this version of the library is thread-safe or not (`UUID_SYSLOG_THREAD_SAFE` and `uuid::syslog::thread_safe`).

Changed

- Make the library thread-safe (for log messages only) when supported by the platform.

5.1.5 2.1.2 – 2022-02-28

Fix performance on the ESP32.

Fixed

- Include lwIP headers directly so that messages can be sent faster on the ESP32.

5.1.6 2.1.1 – 2022-02-23

Silence irrelevant compiler warnings when building on the ESP32.

Changed

- Silence compiler warnings/errors about `%S` in a format string.

5.1.7 2.1.0 – 2022-01-27

More visibility of the log message queue and control over the rate at which messages are dispatched.

Added

- Function to get the current size of the log message queue.
- Symbols to control the UDP message sending delay:
  - `UUID_SYSLOG_UDP_BASE_MESSAGE_DELAY`
  - `UUID_SYSLOG_UDP_IPV4_ARP_MESSAGE_DELAY`
  - `UUID_SYSLOG_UDP_IPV6_NDP_MESSAGE_DELAY`

  It is inadvisable to change these because UDP packets may be queued and then discarded by the platform when the queue limit is reached.

Changed

- Relax IPv6 scope checking when waiting for a local address to allow either global or local unicast addresses to be used with any global or local unicast destination. Previously they had to be the same type.

5.1.8 2.0.6 – 2021-04-18

Upgrade to PlatformIO 5.

Changed

- Use PlatformIO 5 dependency specification.

5.1.9 2.0.5 – 2021-01-17

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

Changed

- Don't unregister handler explicitly in the destructor, this is now handled by the logging library.

5.1.10 2.0.4 – 2019-09-22

Fix log message transmit retries.

Fixed

- Log messages that failed to be sent are not left on the queue correctly and may cause a crash when they are retried.
- Add memory barrier around checks for log queue overflow.

5.1.11 2.0.3 – 2019-09-21

Feature detection fixes.

Fixed

- Use `gettimeofday()` on the ESP32.
- Don't redefine `UUID_SYSLOG_ARP_CHECK`.

5.1.12 2.0.2 – 2019-09-20

Support IPv6 addresses.

Fixed

- Use move constructors on rvalues.
- Support for IPv6 addresses.

5.1.13 2.0.1 – 2019-09-05

Bug fix for trace level messages.

Fixed

- Use debug level for trace level messages.

5.1.14 2.0.0 – 2019-09-03

Additional features and API changes.

Added

- Functions to get all of the configuration parameters.
- Support for sending a `-- MARK --` message when there is no activity for a configurable period of time.

Changed

- Add log level to the message text.
- Lower the log level of log level change messages to `INFO`.
- Rename `set_host()` to `destination()`.
- Remove `get_` and `set_` from function names.

Fixed

- Function to set the maximum number of log messages is no longer limited to 0 or 1.

5.1.15 1.0.0 – 2019-09-01

First stable release.

Added

- Buffer messages during startup until configuration is provided.
- Automatic use of system time if it is available.
- Wait for the network to be available before transmitting.
- Explicit ARP check for the destination host on ESP8266 and ESP32.
- Rate limiting of output packets to avoid dropped messages.
- Configurable destination host and log level.
- Configurable local hostname.
- Configurable queue size.