Machine Translated by Google

RT-THREAD ULOG Logging Component Application Notes - Advanced Edition

RT-THREAD Document Center

Copyright **@2019** Shanghai Ruisaide Electronic Technology Co., Ltd.

WWW.RT-THREAD.ORG

Tuesday 9th October, 2018

Machine Translated by Google

Table of contents

Machine Translated by Google

Machine Translated by Google

This application note, based on the "RT-Thread ulog Logging Component Application Note - Basics," explains the advanced usage and techniques of the RT-Thread ulog component. It helps developers gain a deeper understanding of ulog and improve log debugging efficiency.

1 Purpose and structure of this paper

1.1 Purpose and Background of this Paper

After reading the "RT-Thread ulog Logging Component Application Note - Basics," you'll have a basic understanding of ulog's functionality. To help you better master ulog, this application note will focus on introducing ulog's advanced features and some log debugging tips and tricks. Mastering these advanced techniques will significantly improve your log debugging efficiency.

At the same time, we will also introduce the advanced mode of ulog: syslog mode, which can achieve everything from front-end API to log format Full compatibility with Linux syslog greatly facilitates software migration from Linux.

1.2 Structure of this paper

This application note will introduce the advanced applications of RT-Thread ulog from the following aspects:

• ulog backend • ulog asynchronous mode • ulog log filter • ulog syslog mode • some log debugging tips

2 Problem Description

This application note will introduce the RT-Thread ulog component around the following issues.

• What backends does ulog support? • How to use asynchronous mode, log filter and syslog mode? • How to handle system exceptions (for example, hardfault)? • How to output more intuitive logs?

To solve these problems, you need to have a certain understanding of the advanced functions of the RT-Thread ulog component. At the same time, combined with actual routine hands-on experiments, various functions will also be demonstrated on the qemu platform.

Machine Translated by Google

3. Problem Solving

3.1 Log Backend

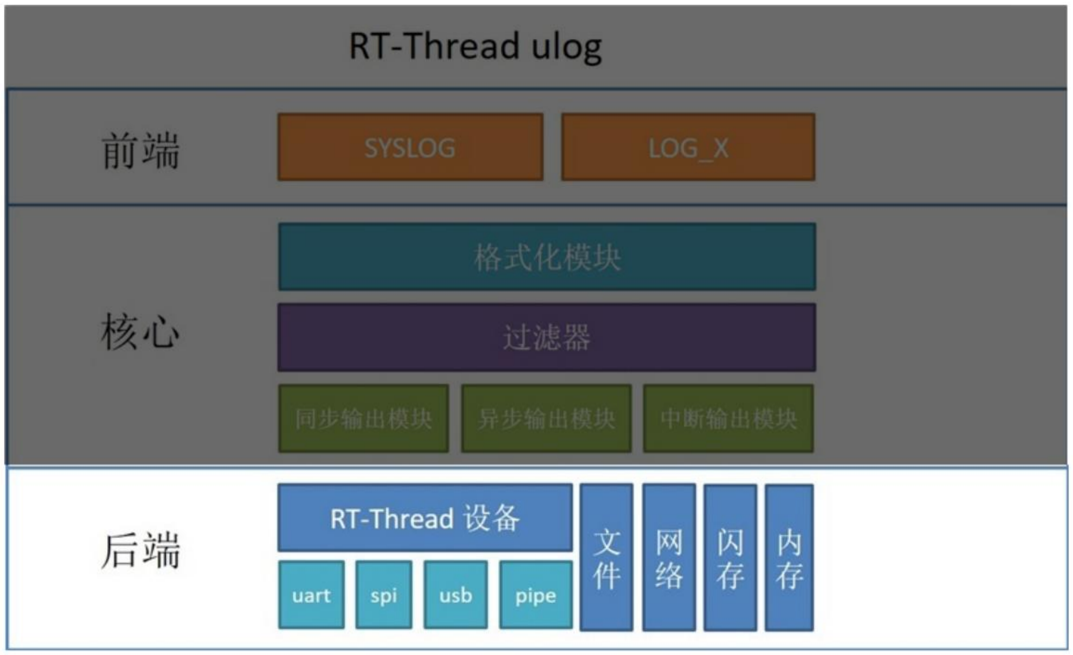


Figure **1:** *ulog* frame

Speaking of backends, let's review the ulog framework diagram. As you can see from the diagram above, ulog uses a frontend-to-backend separation design, with no dependencies between the frontend and backend. Furthermore, it supports a variety of backends; any backend can be registered as long as it's implemented.

Currently, ulog has integrated a console backend, which is the traditional device for outputting rt_kprintf print logs. In the future, ulog will also add the implementation of file backends, Flash backends, network backends, and other backends. Of course, if there are special needs, users can also implement their own backends. The following uses the console backend as an example to briefly introduce the backend implementation and registration method.

Open the rt-thread/components/utilities/ulog/backend/console_be.c file and you can see the

The following contents are included:

```
#include <rthw.h>
#include <ulog.h>

/* Define console backend device*/
static struct ulog_backend console; /* Console backend
output function*/
void ulog_console_backend_output(struct ulog_backend *backend,
        rt_uint32_t level, const char *tag, rt_bool_t is_raw, const char *log , size_t len)

{
```

Machine Translated by Google

```
        ...
        /* Output log to console */
        ...

} /* Console backend initialization*/
int ulog_console_backend_init(void) {

        /* Set output function */
        console.output = ulog_console_backend_output; /* Register
        backend*/
        ulog_backend_register(&console, "console", RT_TRUE);

        return 0;
}
INIT_COMPONENT_EXPORT(ulog_console_backend_init);
```

From the above code, we can see that the implementation of the console backend is very simple. Here we implement the output of the backend device function, and register the backend to ulog, and then all ulog logs will be output to the console.

If you want to implement a more complex backend device, you need to understand the backend device structure, as follows:

```
struct ulog_backend {

        char name[RT_NAME_MAX];
        rt_bool_t support_color; void
        (*init) (struct ulog_backend *backend); void (*output)(struct
        ulog_backend *backend, rt_uint32_t level, const
                char *tag, rt_bool_t is_raw, const char *log, size_t len); void (*flush) (struct
        ulog_backend *backend); void (*deinit)(struct ulog_backend
        *backend); rt_slist_t list;

};
```

From the perspective of this structure, the requirements for implementing backend devices are as follows:

• name and support_color attributes can be passed in during registration via the ulog_backend_register function; • output is the specific output function of the backend, and all backends must implement the interface; • init/deinit are optional, init will be called during register, and deinit will be called during ulog_deinit use;

• Flush is also optional. Some backends with internal output buffering must implement this interface, such as file systems with RAM cache. Backend flushing is generally called by ulog_flush in the event of an assertion, hardfault, or other exception.

Machine Translated by Google

3.2 Asynchronous Logging

In ulog, the default output mode is synchronous mode, but in many scenarios, users may also need asynchronous mode. When users call the log output API, the logs will be cached in the buffer, and a thread dedicated to log output will retrieve the logs and output them to the backend.

3.2.1. Asynchronous Mode vs. Synchronous Mode

For users, there is no difference in the use of the log API between the two modes, because ulog will handle the underlying

There is a difference. The difference in their working principles is roughly shown in the figure below:

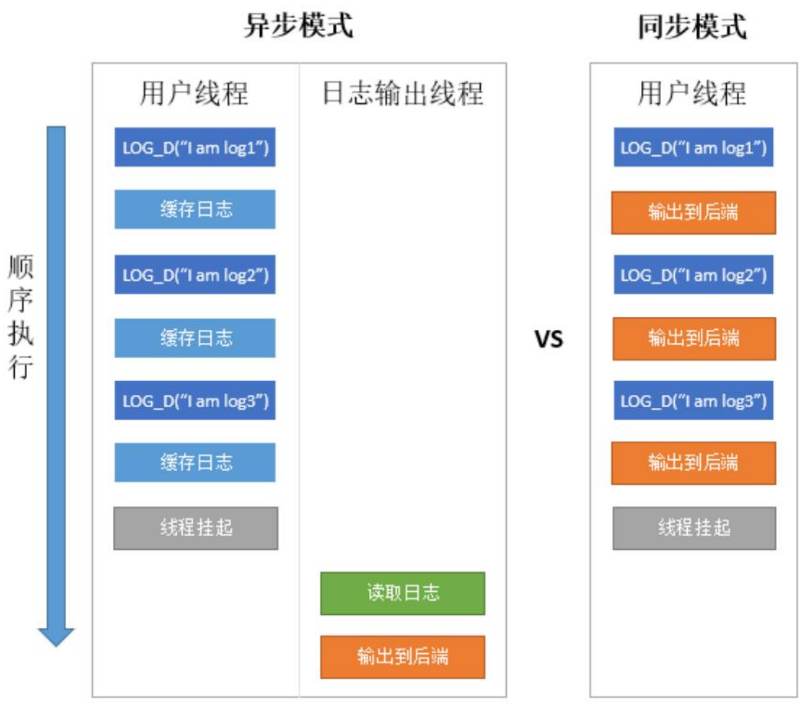


Figure 2: *ulog* asynchronous *VS* synchronous

Let's look at the advantages and disadvantages of asynchronous mode

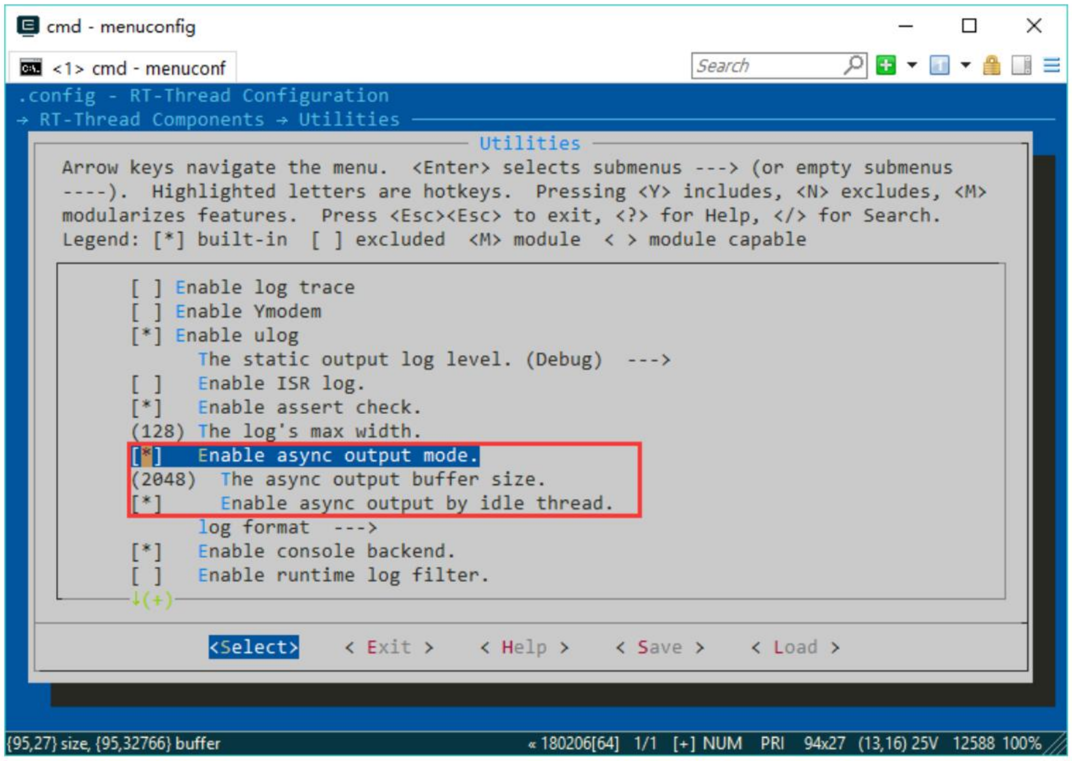
- advantage:
  - **First,** the log output will not block the current thread, and some backend output rates are low, so the same The step output mode may affect the timing of the current thread, but the asynchronous mode does not have this problem.
  - Secondly, since each thread using the log omits the backend output action, the stack of these threads opens Sales may also be reduced, and from this perspective the resource usage of the entire system can be reduced.
  - In synchronous mode, the interruption log can only be output to the console backend, while in asynchronous mode, the interruption log can be output Go to all backends.
- Disadvantages: First, asynchronous mode requires a log buffer. Second, asynchronous log output requires a dedicated thread, such as an idle thread or a user-defined thread, making its use somewhat complex. Overall, asynchronous mode consumes more resources than synchronous mode.

Machine Translated by Google

3.2.2. Configuration of asynchronous mode

Open env, enter the rt-thread\bsp\qemu-vexpress-a9 folder, open menuconfig and find ulog

The asynchronous output mode option, when turned on, has the following effects:



```
.config - RT-Thread Configuration
→ RT-Thread Components → Utilities ─────────────────────────
                          Utilities
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus
  ----).  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M>
  modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

        [ ] Enable log trace
        [ ] Enable Ymodem
        [*] Enable ulog
              The static output log level. (Debug)  --->
        [ ]   Enable ISR log.
        [*]   Enable assert check.
        (128) The log's max width.
        [*]   Enable async output mode.
        (2048)  The async output buffer size.
        [*]     Enable async output by idle thread.
              log format  --->
        [*]   Enable console backend.
        [ ]   Enable runtime log filter.
        ↓(+)

     <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 3: *ulog* Asynchronous configuration

For asynchronous mode, there are two options

• The async output buffer size. : The asynchronous buffer size, defaults to 2048. • Enable async output by idle thread. :

Whether to let the idle thread as the asynchronous log output thread, so that when the system is idle, it can automatically complete the log output. This option is enabled by default. If you want to modify other threads, please turn this option off first, and then call the ulog_async_output() function in a new thread in a loop.

Note: When using idle thread output, be sure to ensure that the idle thread stack size is at least 384 bytes.

3.2.3. Asynchronous Mode Routines

• Save the asynchronous output option

configuration • Copy rt-thread\examples\ulog_example.c to rt-thread\bsp\qemu-vexpress-a9\

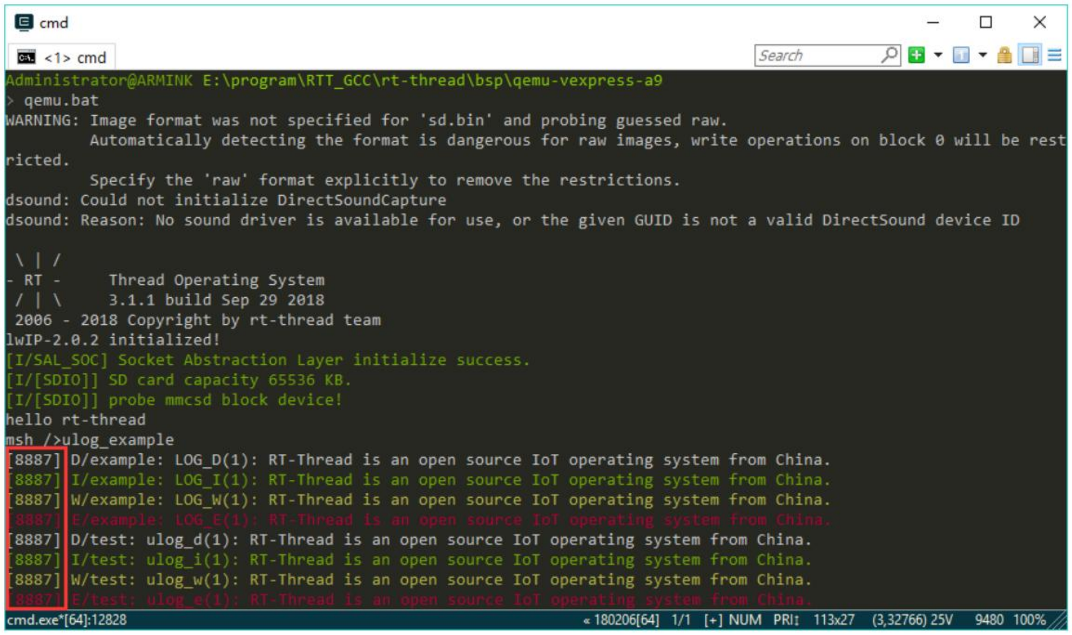
In the applications folder, execute the

scons command and wait for the compilation to complete. Run

qemu.bat to open the RT-Thread qemu simulator. Enter the ulog_example command to see the

results of the ulog routine. The general effect is as shown below.

Machine Translated by Google



```
Administrator@ARMINK E:\program\RTT_GCC\rt-thread\bsp\qemu-vexpress-a9
> qemu.bat
WARNING: Image format was not specified for 'sd.bin' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be rest
ricted.
         Specify the 'raw' format explicitly to remove the restrictions.
dsound: Could not initialize DirectSoundCapture
dsound: Reason: No sound driver is available for use, or the given GUID is not a valid DirectSound device ID

 \ | /
- RT -     Thread Operating System
 / | \     3.1.1 build Sep 29 2018
 2006 - 2018 Copyright by rt-thread team
lwIP-2.0.2 initialized!
[I/SAL_SOC] Socket Abstraction Layer initialize success.
[I/[SDIO]] SD card capacity 65536 KB.
[I/[SDIO]] probe mmcsd block device!
hello rt-thread
msh />ulog_example
[8887] D/example: LOG_D(1): RT-Thread is an open source IoT operating system from China.
[8887] I/example: LOG_I(1): RT-Thread is an open source IoT operating system from China.
[8887] W/example: LOG_W(1): RT-Thread is an open source IoT operating system from China.
[8887] E/example: LOG_E(1): RT-Thread is an open source IoT operating system from China.
[8887] D/test: ulog_d(1): RT-Thread is an open source IoT operating system from China.
[8887] I/test: ulog_i(1): RT-Thread is an open source IoT operating system from China.
[8887] W/test: ulog_w(1): RT-Thread is an open source IoT operating system from China.
[8887] E/test: ulog_e(1): RT-Thread is an open source IoT operating system from China.
```

cmd
<1> cmd
Search
cmd.exe*[64]:12828
« 180206[64] 1/1 [+] NUM PRI↕ 113x27 (3,32766) 25V 9480 100%

Figure 4: *ulog* Asynchronous routines

A careful observation will reveal that after enabling asynchronous mode, the time information for these logs, which appear very close together in the code, is almost identical. However, in synchronous mode, logs are output using user threads. Since log output takes time, there is a certain gap between each log entry. This clearly demonstrates the high efficiency of asynchronous log output, which takes almost no time from the caller.

3.3 Log Filter (Dynamic Filtering)

In the basics of the ulog application note, some static filtering functions of logs are introduced. Static filtering has its advantages, such as saving resources. However, in many cases, users need to dynamically adjust the log filtering method during software runtime. This is where the dynamic filter function of ulog can be used. To use the dynamic filter function, you need to enable the Enable runtime log filter option in menuconfig. This option is disabled by default.

Ulog supports the following four dynamic filtering methods, and each has corresponding API functions and Finsh/MSH commands, which will be introduced one by one below.

3.3.1. Filtering by module level

- **Function: int** ulog_tag_lvl_filter_set(const **char** *tag, rt_uint32_t level) • Command format: ulog_tag_lvl <tag> <level>

The modules mentioned here are introduced in the basics of the application note and represent a class of logging code with the same tag attributes. Sometimes, you need to dynamically change the log output level of a module at runtime. For example, to disable logging for the Wi-Fi module, you can set the log output level of the Wi-Fi module to LOG_FILTER_LVL_SILENT .

Machine Translated by Google

3.3.2. Global filtering by level

• **Function: void** ulog_global_filter_lvl_set(rt_uint32_t level) • Command format: ulog_lvl <level> , level is

- 0: Assert
- 3: Error
- 4: Warning
- 6: Info
- 7: Debug

After setting the global filtering level through functions or commands, logs below the set level will stop being output.

3.3.3. Global filtering by tag

• **Function: void** ulog_global_filter_tag_set(const **char** *tag) • Command format: ulog_tag [tag] , when tag is empty, tag filtering is canceled

This filtering method can filter all logs by tags, and only logs containing tag information are allowed to be output.

For example, if there are logs with the tags wifi.driver , wifi.mgnt , and audio.driver , and the filter tag is set to wifi , only logs with the tags wifi.driver and wifi.mgnt will be output. Similarly, if the filter tag is set to driver , only logs with the tags wifi.driver and audio.driver will be output.

3.3.4. Global filtering by keyword

• **Function: void** ulog_global_filter_kw_set(const **char** *keyword) • Command format: ulog_kw [keyword] , when keyword is empty, keyword filtering is canceled

This filtering method can filter all logs by keywords, and only logs containing keyword information are allowed to be output.

3.3.5. Run the routine

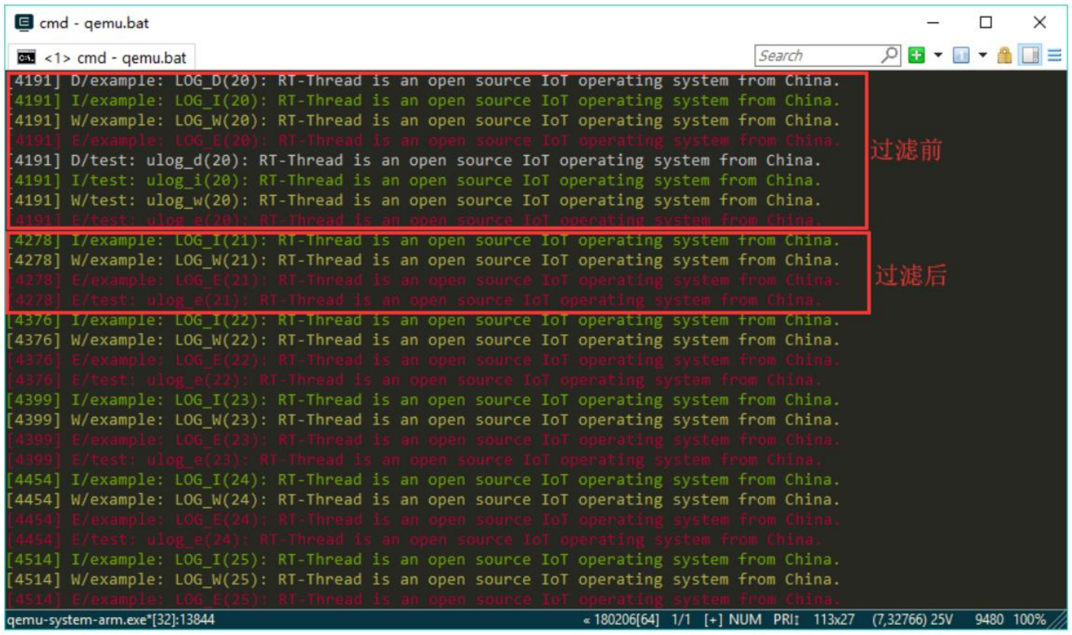
Still executing in qemu BSP, first enable dynamic filtering in menuconfig, then save the configuration and compile, When the routine is run, after the log is output about **20** times, the corresponding filtering code in ulog_example.c will be executed:

```
if (count == 20) {

    /* Set the global filer level is INFO. All of DEBUG log will stop
        output */
    ulog_global_filter_lvl_set(LOG_LVL_INFO);
    /* Set the test tag's level filter's level is ERROR. The DEBUG, INFO,
         WARNING log will stop output. */
    ulog_tag_lvl_filter_set("test", LOG_LVL_ERROR);
}
...
```

Machine Translated by Google

At this point, the global filtering level is set to INFO, so logs with a level lower than INFO are no longer visible. At the same time, the log output level for the test tag is set to ERROR, and all logs with a level lower than ERROR in the test tag are no longer output. Each log entry has a count of the number of times the log has been output. The comparison is as follows:



```
[4191] D/example: LOG_D(20): RT-Thread is an open source IoT operating system from China.
[4191] I/example: LOG_I(20): RT-Thread is an open source IoT operating system from China.
[4191] W/example: LOG_W(20): RT-Thread is an open source IoT operating system from China.
[4191] E/example: LOG_E(20): RT-Thread is an open source IoT operating system from China.
[4191] D/test: ulog_d(20): RT-Thread is an open source IoT operating system from China.
[4191] I/test: ulog_i(20): RT-Thread is an open source IoT operating system from China.
[4191] W/test: ulog_w(20): RT-Thread is an open source IoT operating system from China.
[4191] E/test: ulog_e(20): RT-Thread is an open source IoT operating system from China.
[4278] I/example: LOG_I(21): RT-Thread is an open source IoT operating system from China.
[4278] W/example: LOG_W(21): RT-Thread is an open source IoT operating system from China.
[4278] E/example: LOG_E(21): RT-Thread is an open source IoT operating system from China.
[4278] E/test: ulog_e(21): RT-Thread is an open source IoT operating system from China.
[4376] I/example: LOG_I(22): RT-Thread is an open source IoT operating system from China.
[4376] W/example: LOG_W(22): RT-Thread is an open source IoT operating system from China.
[4376] E/example: LOG_E(22): RT-Thread is an open source IoT operating system from China.
[4376] E/test: ulog_e(22): RT-Thread is an open source IoT operating system from China.
[4399] I/example: LOG_I(23): RT-Thread is an open source IoT operating system from China.
[4399] W/example: LOG_W(23): RT-Thread is an open source IoT operating system from China.
[4399] E/example: LOG_E(23): RT-Thread is an open source IoT operating system from China.
[4399] E/test: ulog_e(23): RT-Thread is an open source IoT operating system from China.
[4454] I/example: LOG_I(24): RT-Thread is an open source IoT operating system from China.
[4454] W/example: LOG_W(24): RT-Thread is an open source IoT operating system from China.
[4454] E/example: LOG_E(24): RT-Thread is an open source IoT operating system from China.
[4454] E/test: ulog_e(24): RT-Thread is an open source IoT operating system from China.
[4514] I/example: LOG_I(25): RT-Thread is an open source IoT operating system from China.
[4514] W/example: LOG_W(25): RT-Thread is an open source IoT operating system from China.
[4514] E/example: LOG_E(25): RT-Thread is an open source IoT operating system from China.
qemu-system-arm.exe*[32]:13844        « 180206[64]  1/1  [+] NUM  PRI:  113x27  (7,32766) 25V  9480 100%
```

Figure **5:** *ulog* Filter routines *20*

After the log is output about **30** times, the following filtering code in ulog_example.c will be executed:

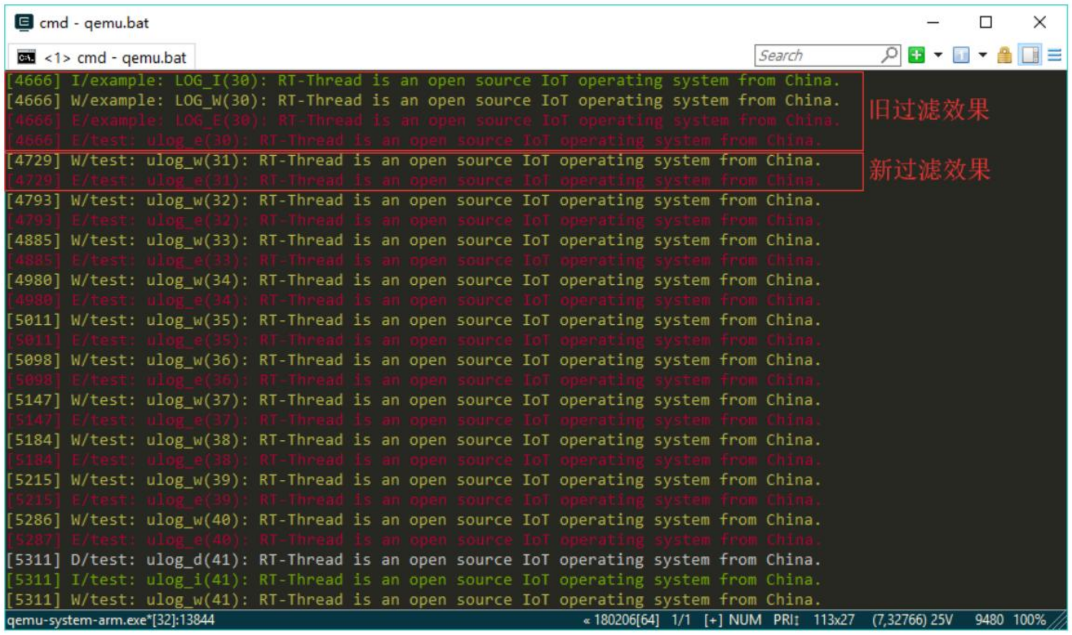
```
...
else if (count == 30) {

    /* Set the example tag's level filter's level is
        LOG_FILTER_LVL_SILENT, the log enter silent mode. */
    ulog_tag_lvl_filter_set("example", LOG_FILTER_LVL_SILENT); /* Set the test tag's
    level filter's level is WARNING. The DEBUG, INFO log will stop output. */

    ulog_tag_lvl_filter_set("test", LOG_LVL_WARNING);
}
...
```

Now we've added a filter for the example module and stopped all log output for this module, so we won't see any logs from this module. At the same time, we've lowered the log output level for the test tag to WARING, so that we can only see WARING and ERROR level logs for the test tag. The effect is as follows:

Machine Translated by Google



```
[4666] I/example: LOG_I(30): RT-Thread is an open source IoT operating system from China.
[4666] W/example: LOG_W(30): RT-Thread is an open source IoT operating system from China.
[4666] E/example: LOG_E(30): RT-Thread is an open source IoT operating system from China.
[4666] E/test: ulog_e(30): RT-Thread is an open source IoT operating system from China.
[4729] W/test: ulog_w(31): RT-Thread is an open source IoT operating system from China.
[4729] E/test: ulog_e(31): RT-Thread is an open source IoT operating system from China.
[4793] W/test: ulog_w(32): RT-Thread is an open source IoT operating system from China.
[4793] E/test: ulog_e(32): RT-Thread is an open source IoT operating system from China.
[4885] W/test: ulog_w(33): RT-Thread is an open source IoT operating system from China.
[4885] E/test: ulog_e(33): RT-Thread is an open source IoT operating system from China.
[4980] W/test: ulog_w(34): RT-Thread is an open source IoT operating system from China.
[4980] E/test: ulog_e(34): RT-Thread is an open source IoT operating system from China.
[5011] W/test: ulog_w(35): RT-Thread is an open source IoT operating system from China.
[5011] E/test: ulog_e(35): RT-Thread is an open source IoT operating system from China.
[5098] W/test: ulog_w(36): RT-Thread is an open source IoT operating system from China.
[5098] E/test: ulog_e(36): RT-Thread is an open source IoT operating system from China.
[5147] W/test: ulog_w(37): RT-Thread is an open source IoT operating system from China.
[5147] E/test: ulog_e(37): RT-Thread is an open source IoT operating system from China.
[5184] W/test: ulog_w(38): RT-Thread is an open source IoT operating system from China.
[5184] E/test: ulog_e(38): RT-Thread is an open source IoT operating system from China.
[5215] W/test: ulog_w(39): RT-Thread is an open source IoT operating system from China.
[5215] E/test: ulog_e(39): RT-Thread is an open source IoT operating system from China.
[5286] W/test: ulog_w(40): RT-Thread is an open source IoT operating system from China.
[5287] E/test: ulog_e(40): RT-Thread is an open source IoT operating system from China.
[5311] D/test: ulog_d(41): RT-Thread is an open source IoT operating system from China.
[5311] I/test: ulog_i(41): RT-Thread is an open source IoT operating system from China.
[5311] W/test: ulog_w(41): RT-Thread is an open source IoT operating system from China.
qemu-system-arm.exe*[32]:13844          « 180206[64]  1/1  [+] NUM  PRI↕  113x27  (7,32766) 25V  9480 100%
```

Figure **6:** *ulog* Filter routines *30*

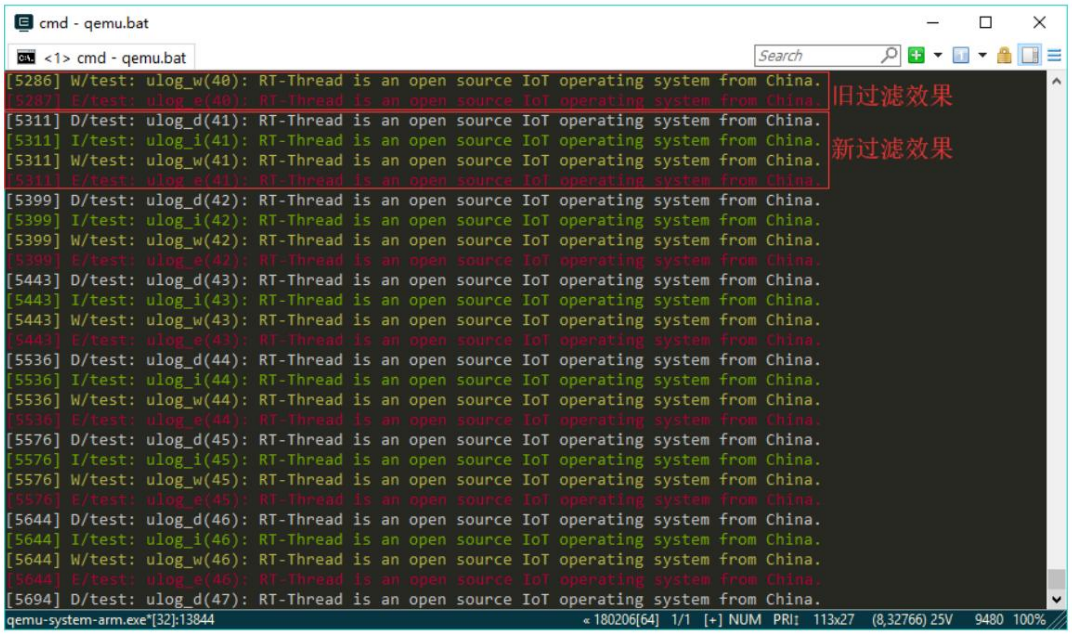
After the log is output about **40** times, the following filter code in ulog_example.c will be executed:

```
...
else if (count == 40) {

    /* Set the test tag's level filter's level is LOG_FILTER_LVL_ALL. All
          level log will resume output. */
    ulog_tag_lvl_filter_set("test", LOG_FILTER_LVL_ALL);
    /* Set the global filer level is LOG_FILTER_LVL_ALL. All level log
        will resume output */
    ulog_global_filter_lvl_set(LOG_FILTER_LVL_ALL);
}
```

At this point, the log output level of the test module is adjusted to LOG_FILTER_LVL_ALL , which means that logs of any level in this module will no longer be filtered. At the same time, the global filter level is set to LOG_FILTER_LVL_ALL , so all logs of the test module will be restored. The effect is as follows:

Machine Translated by Google


cmd - qemu.bat
<1> cmd - qemu.bat
Search
旧过滤效果
新过滤效果
qemu-system-arm.exe*[32]:13844
« 180206[64] 1/1 [+] NUM PRI↕ 113x27 (8,32766) 25V 9480 100%

```
[5286] W/test: ulog_w(40): RT-Thread is an open source IoT operating system from China.
[5287] E/test: ulog_e(40): RT-Thread is an open source IoT operating system from China.
[5311] D/test: ulog_d(41): RT-Thread is an open source IoT operating system from China.
[5311] I/test: ulog_i(41): RT-Thread is an open source IoT operating system from China.
[5311] W/test: ulog_w(41): RT-Thread is an open source IoT operating system from China.
[5311] E/test: ulog_e(41): RT-Thread is an open source IoT operating system from China.
[5399] D/test: ulog_d(42): RT-Thread is an open source IoT operating system from China.
[5399] I/test: ulog_i(42): RT-Thread is an open source IoT operating system from China.
[5399] W/test: ulog_w(42): RT-Thread is an open source IoT operating system from China.
[5399] E/test: ulog_e(42): RT-Thread is an open source IoT operating system from China.
[5443] D/test: ulog_d(43): RT-Thread is an open source IoT operating system from China.
[5443] I/test: ulog_i(43): RT-Thread is an open source IoT operating system from China.
[5443] W/test: ulog_w(43): RT-Thread is an open source IoT operating system from China.
[5443] E/test: ulog_e(43): RT-Thread is an open source IoT operating system from China.
[5536] D/test: ulog_d(44): RT-Thread is an open source IoT operating system from China.
[5536] I/test: ulog_i(44): RT-Thread is an open source IoT operating system from China.
[5536] W/test: ulog_w(44): RT-Thread is an open source IoT operating system from China.
[5536] E/test: ulog_e(44): RT-Thread is an open source IoT operating system from China.
[5576] D/test: ulog_d(45): RT-Thread is an open source IoT operating system from China.
[5576] I/test: ulog_i(45): RT-Thread is an open source IoT operating system from China.
[5576] W/test: ulog_w(45): RT-Thread is an open source IoT operating system from China.
[5576] E/test: ulog_e(45): RT-Thread is an open source IoT operating system from China.
[5644] D/test: ulog_d(46): RT-Thread is an open source IoT operating system from China.
[5644] I/test: ulog_i(46): RT-Thread is an open source IoT operating system from China.
[5644] W/test: ulog_w(46): RT-Thread is an open source IoT operating system from China.
[5644] E/test: ulog_e(46): RT-Thread is an open source IoT operating system from China.
[5694] D/test: ulog_d(47): RT-Thread is an open source IoT operating system from China.
```

Figure **7:** *ulog* Filter routines *40*

3.4 Usage when the system is abnormal

Because ulog's asynchronous mode has a cache mechanism, the registered backend may also have an internal cache. If the system encounters an error such as a hardfault or assertion, but there are logs in the cache that have not been output, this may cause log loss and make it difficult to find the cause of the exception.

For this scenario, ulog provides a unified log flush **function: void** ulog_flush(void) . When an exception occurs, the exception information log is output and the function is called at the same time to ensure that the remaining logs in the cache can also be output to the backend.

The following example uses RT-Thread assertions and CmBacktrace:

3.4.1. Assertion

RT-Thread's assertion supports assertion callbacks (hook). We define an assertion hook function similar to the following, and then Then set it to the system through the rt_assert_set_hook(rtt_user_assert_hook); function.

```
static void rtt_user_assert_hook(const char* ex, const char* func,
         rt_size_t line)
{
          rt_enter_critical();

          ulog_output(LOG_LVL_ASSERT,"rtt", "(%s) has assert failed at %s:%ld." , ex, func, line);

          /* flush all log */ ulog_flush();
          while(1);
```

Machine Translated by Google

```
}
```

3.4.2. CmBacktrace

CmBacktrace is an error diagnosis library for ARM Cortex-M series MCUs. It also has a corresponding RT-Thread software package, and the latest version of the software package has been adapted for ulog. The adaptation code is located in cmb_cfg.h :

```
...
/* print line, must config by user */
#include <rtthread.h>
#ifndef RT_USING_ULOG
#define cmb_println(...) ("\r\n")                    rt_kprintf(__VA_ARGS__);rt_kprintf

#else
#include <ulog.h>
#define cmb_println(...)                             ulog_e("cmb", __VA_ARGS__);
        ulog_flush()
#endif /* RT_USING_ULOG */
...
```

From this we can see that when ulog is enabled, each log output of CmBacktrace will use error level, and ulog_flush will be executed at the same time , and the user does not need to make any further modifications.

3.5 Syslog Mode

On Unix-like operating systems, syslog is widely used for system logs. Common backends for syslog include files and networks. The syslog log can be recorded in a local file or sent to a syslog receiving server over the network.

ulog provides support for syslog mode. Not only is the front-end API completely consistent with the syslog API, but the log format also complies with the RFC standard. However, it should be noted that after turning on syslog mode, no matter which log output API is used, the entire ulog log output format will adopt the syslog format.

3.5.1. Syslog Configuration

Just turn on the option Enable syslog format log and API.

Machine Translated by Google

3.5.2. Log Format

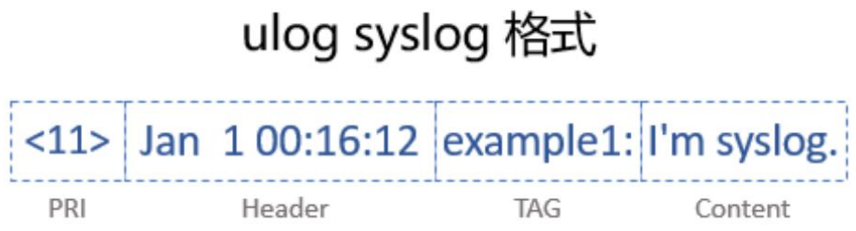


Figure 8: *ulog syslog* Format

As shown in the figure above, the ulog syslog log format is divided into the following four parts:

• **PRI :** The PRI part consists of a number enclosed in angle brackets. This number contains the program module (Facility) and severity (Severity) information. It is obtained by multiplying the Facility by 8 and then adding the Severity. Facility and Severity are passed in as parameters to the syslog function. For specific values, see syslog.h; • **Header :** The Header part is mainly a timestamp, indicating the time of the current log; • **TAG :** The tag of the current log, which can be passed in as a parameter to the openlog function. If not specified, rtt will be used. As the default label;

• **Content :** The specific content of the log.

3.5.3. Usage

Before use, you need to enable the syslog option in menuconfig. The main commonly used APIs are:

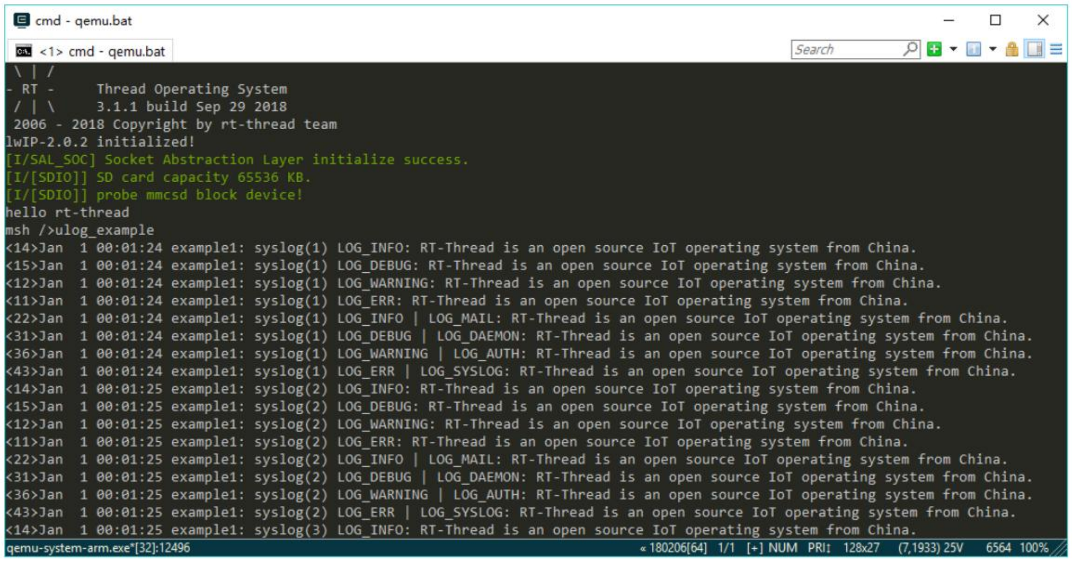
• Open syslog: void openlog(const char *ident, int option, int facility)

• Output syslog log: void syslog(int priority, const char *format, …)

Tip: Calling openlog is optional. If you do not call openlog, it will be automatically called when syslog is called for the first time.

The usage of syslog() function is also very simple, and its input format is the same as printf function. There are also syslog routines in qemu. The running effect in qemu is as follows:

Machine Translated by Google



```
 \ | /
- RT -     Thread Operating System
 / | \     3.1.1 build Sep 29 2018
 2006 - 2018 Copyright by rt-thread team
lwIP-2.0.2 initialized!
[I/SAL_SOC] Socket Abstraction Layer initialize success.
[I/[SDIO]] SD card capacity 65536 KB.
[I/[SDIO]] probe mmcsd block device!
hello rt-thread
msh />ulog_example
<14>Jan  1 00:01:24 example1: syslog(1) LOG_INFO: RT-Thread is an open source IoT operating system from China.
<15>Jan  1 00:01:24 example1: syslog(1) LOG_DEBUG: RT-Thread is an open source IoT operating system from China.
<12>Jan  1 00:01:24 example1: syslog(1) LOG_WARNING: RT-Thread is an open source IoT operating system from China.
<11>Jan  1 00:01:24 example1: syslog(1) LOG_ERR: RT-Thread is an open source IoT operating system from China.
<22>Jan  1 00:01:24 example1: syslog(1) LOG_INFO | LOG_MAIL: RT-Thread is an open source IoT operating system from China.
<31>Jan  1 00:01:24 example1: syslog(1) LOG_DEBUG | LOG_DAEMON: RT-Thread is an open source IoT operating system from China.
<36>Jan  1 00:01:24 example1: syslog(1) LOG_WARNING | LOG_AUTH: RT-Thread is an open source IoT operating system from China.
<43>Jan  1 00:01:24 example1: syslog(1) LOG_ERR | LOG_SYSLOG: RT-Thread is an open source IoT operating system from China.
<14>Jan  1 00:01:25 example1: syslog(2) LOG_INFO: RT-Thread is an open source IoT operating system from China.
<15>Jan  1 00:01:25 example1: syslog(2) LOG_DEBUG: RT-Thread is an open source IoT operating system from China.
<12>Jan  1 00:01:25 example1: syslog(2) LOG_WARNING: RT-Thread is an open source IoT operating system from China.
<11>Jan  1 00:01:25 example1: syslog(2) LOG_ERR: RT-Thread is an open source IoT operating system from China.
<22>Jan  1 00:01:25 example1: syslog(2) LOG_INFO | LOG_MAIL: RT-Thread is an open source IoT operating system from China.
<31>Jan  1 00:01:25 example1: syslog(2) LOG_DEBUG | LOG_DAEMON: RT-Thread is an open source IoT operating system from China.
<36>Jan  1 00:01:25 example1: syslog(2) LOG_WARNING | LOG_AUTH: RT-Thread is an open source IoT operating system from China.
<43>Jan  1 00:01:25 example1: syslog(2) LOG_ERR | LOG_SYSLOG: RT-Thread is an open source IoT operating system from China.
<14>Jan  1 00:01:25 example1: syslog(3) LOG_INFO: RT-Thread is an open source IoT operating system from China.
qemu-system-arm.exe*[32]:12496                « 180206[64]  1/1  [+] NUM  PRI↕  128x27  (7,1933) 25V  6564 100%
```

Figure **9:** *ulog syslog* Routines

3.6 How to output more intuitive logs

Even with logging tools, improper use can lead to issues like log abuse and a failure to highlight key points. Here, we'll share some tips on using logging components to make log information more intuitive. The main points of interest are:

3.6.1. Log tag classification

Reasonable use of the label function, each module code before using the log, first clearly identify the module, sub-module name. It allows logs to be classified at the very beginning and prepares for later log filtering.

3.6.2. Reasonable use of log levels

When first using the logging library, you'll often encounter issues such as being unable to distinguish between warning and error logs, or between information and debug logs, leading to inappropriate log level selection. Important logs may be invisible, while unimportant logs are everywhere. Therefore, before using the library, be sure to carefully read the log level section in the "RT-Thread ulog Logging Component Application Note - Basics" for clear standards for each level.

3.6.3. Avoiding redundant logs

In some cases, code may be repeatedly called or executed in a loop, resulting in the output of identical or similar log files. This log file not only consumes significant system resources but also hinders developers' ability to troubleshoot issues. Therefore, in such cases, it is recommended to implement special handling for repetitive logs. For example, you can have the upper layer output business-related logs, while the lower layer only returns specific results. For identical log files at the same time, can duplicate log files be deduplicated, with the log file outputted only once if the error status remains unchanged?

Machine Translated by Google

3.6.4. Enable more log formats

The default ulog log format does not include timestamps and thread information. These two log features are quite useful on RTOS. They help developers intuitively understand the execution time and time difference of each log, and clearly identify the thread in which the current code is executed. Therefore, if conditions permit, it is recommended to enable them.

3.6.5. Turn off unimportant logs

Ulog provides multiple dimensions of log switches and filtering functions, which can achieve fine-grained control. Therefore, when debugging a functional module, you can appropriately turn off the log output of other irrelevant modules, so that you can focus on the module currently being debugged. For more information about log filtering functions, please read the "RT-Thread ulog Log Component Application Note - Basics" setting level classification section and the log filter section of this application note.

4 Frequently Asked Questions

- 1. A warning message appears during operation: Warning: There is no enough buffer for saving async log, please increase the ULOG_ASYNC_OUTPUT_BUF_SIZE option.

  When you encounter this prompt, it indicates that the buffer overflow has occurred in asynchronous mode, which will cause some logs to be lost. Increasing the ULOG_ASYNC_OUTPUT_BUF_SIZE option can solve this problem.

- 2. Compile-time prompt: The idle thread stack size must be more than 384 when using async output by idle (ULOG_ASYNC_OUTPUT_BY_IDLE)

  When using an idle thread as an output thread, the stack size of the idle thread needs to be increased, which also depends on the specific backend device. For example, for a console backend, the idle thread must have at least 384 bytes.

5References

5.1 All related APIs in this article

API List

API	Location
rt_err_t ulog_backend_register(ulog_backend_t backend, const char *name, rt_bool_t support_color)	ulog.c
void ulog_async_output(void)	ulog.c
int ulog_tag_lvl_filter_set(const char *tag, rt_uint32_t level)	ulog.c
void ulog_global_filter_lvl_set(rt_uint32_t level)	ulog.c
void ulog_global_filter_tag_set(const char *tag)	ulog.c

Machine Translated by Google

API	Location
void ulog_global_filter_kw_set(const char *keyword)	ulog.c
void openlog(const char *ident, int option, int facility)	syslog.c
void syslog(int priority, const char *format, …)	syslog.c
int setlogmask(int mask)	syslog.c

5.1.2. Detailed explanation of core **API**

5.1.3. Registering backend devices

```
rt_err_t ulog_backend_register(ulog_backend_t backend, const char *name,
        rt_bool_t support_color)
```

Register the backend device to ulog. Before registration, make sure the function members in the backend device structure have been set.

parameter	describe
backend	AT client uses device name
name	AT client supports the maximum length of received data
support_color	Whether to support color log
return	describe
>=0	success

5.1.4. Output all logs in asynchronous mode

```
void ulog_async_output(void)
```

In asynchronous mode, if you want to use other non-idle threads as log output threads, you need to Call the API in a loop to output the logs in the buffer to all backend devices

5.1.5. Filtering logs by module/tag level

```
int ulog_tag_lvl_filter_set(const char *tag, rt_uint32_t level)
```

This API allows you to filter logs by tag level, for example:

```
/* Stop outputting the log of the wifi.driver module*/
ulog_tag_lvl_filter_set("wifi.driver", LOG_FILTER_LVL_SILENT);
/* Stop the output of logs below the INFO level of the wifi.mgnt module*/
```

Machine Translated by Google

```
ulog_tag_lvl_filter_set("wifi.mgnt", LOG_LVL_INFO);
```

parameter	describe
tag	Log tags
level	Set the log level, see ulog_def.h for details
return	describe
>=0	success
-5	Failed, not enough memory

5.1.6. Filtering logs by level (global)

```
void ulog_global_filter_lvl_set(rt_uint32_t level)
```

Set the global log filter level. Logs below this level will stop being output. The levels that can be set include:

level	name
LOG_LVL_ASSERT	assertion
LOG_LVL_ERROR	mistake
LOG_LVL_WARNING	warn
LOG_LVL_INFO	information
LOG_LVL_DBG	debug
LOG_FILTER_LVL_SILENT	Silence (stop output)
LOG_FILTER_LVL_ALL	all

Parameter Description
level The level set by level, see the table above for details, also see ulog_def.h

5.1.7. Filtering Logs by Tags (Globally)

```
void ulog_global_filter_tag_set(const char *tag)
```

Set the global log filter tag. Only when the tag content of the log contains the set string will it be allowed to log.

Output.

Machine Translated by Google

parameter	describe
tag	Set filter tags

5.1.8. Filtering Logs by Keyword (Global)

```
void ulog_global_filter_kw_set(const char *keyword)
```

Set a global log filter keyword, and only logs containing this keyword will be allowed to be output.

parameter	describe
keyword	Set filter keywords