RT-THREAD ULOG Log Component Application Notes **-** Advanced Edition

RT-THREAD Documentation Center

WWW.RT-THREAD.ORG

Tuesday 9th October, 2018

Table of contents

!!! abstract "Abstract" This application note is based on "RT-Thread ulog log component application note - basics", explaining the advanced usage and usage skills of RT-Thread ulog component. It helps developers to have 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 understanding "RT-Thread ulog log component application note - basics", the basic functions of ulog can be mastered. In order to let everyone play ulog better, this application note will focus on introducing ulog's advanced functions and some log debugging experience and skills. After learning these advanced usages, developers can also greatly improve the efficiency of log debugging.

At the same time, we will 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
tips for log debugging

2 Problem Statement

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 should ulog handle
system exceptions (e.g. 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

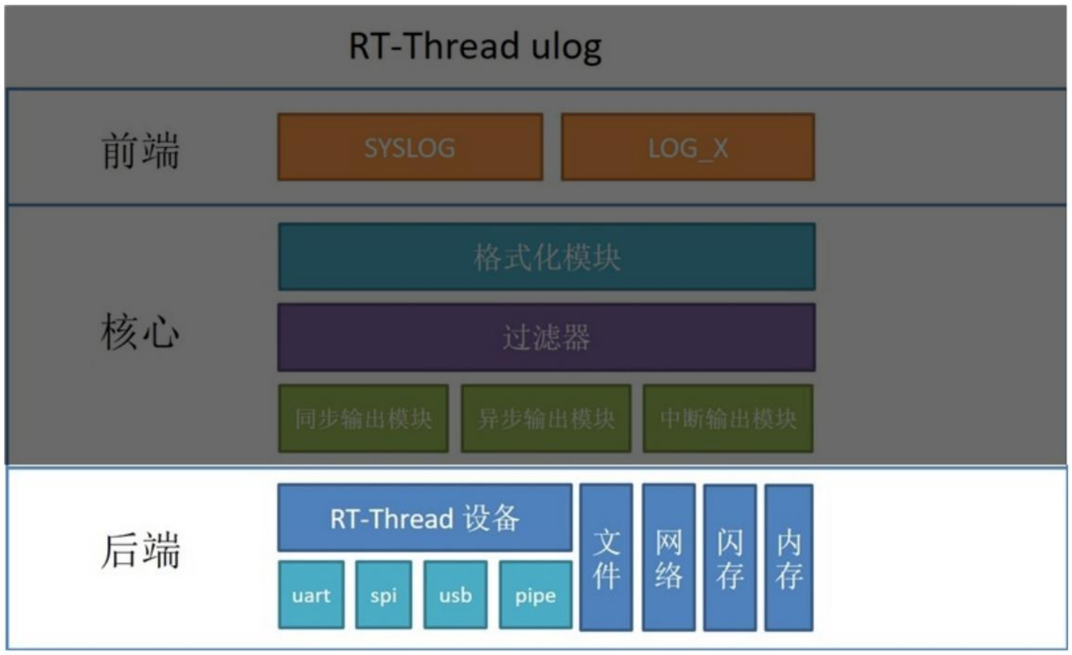


Figure 1: *ulog* frame

Speaking of the backend, let's review the framework diagram of ulog. As can be seen from the above figure, ulog adopts a design that separates the front-end and back-end, and the front-end and back-end have no dependencies. And it supports a variety of backends. No matter what kind of backend, as long as it is implemented, it can be registered.

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

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

The following are the contents:

```
#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)

{
        ...
        /* 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 when registering through the ulog_backend_register function; • output is the specific output function of the backend, and all backends must implement the interface;

• init/deinit can be implemented selectively, init will be called during register, and deinit will be called during ulog_deinit use;

• Flush is also optional. Some backends with internal output caches must implement this interface. For example, some file systems with RAM caches. The backend flush is usually called by ulog_flush in abnormal situations such as assertions and hardfaults.

Machine Translated by Google

3.2 Asynchronous Logging

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

There is a difference. The difference in the working principles of the two is roughly shown in the following figure:

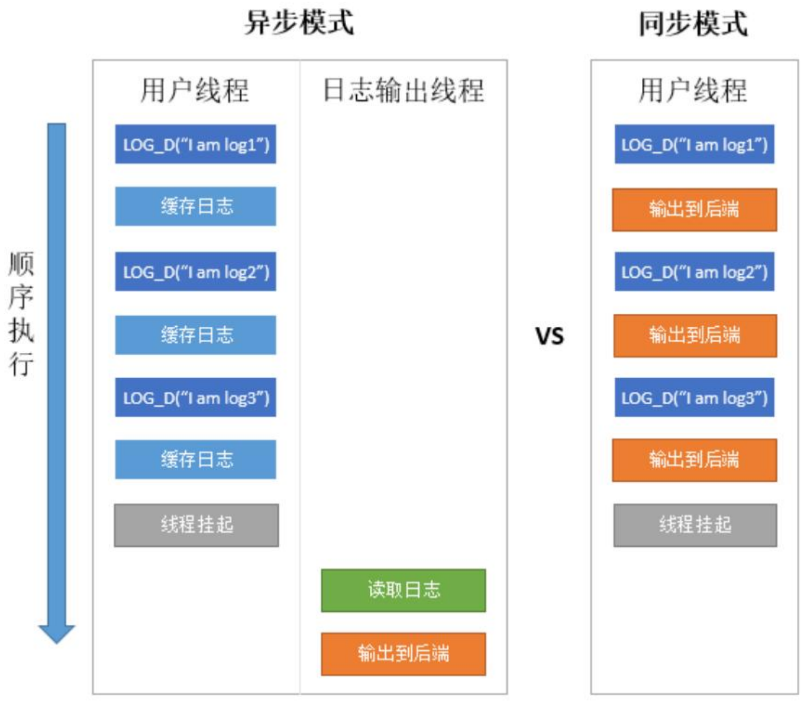


Figure 2: *ulog* asynchronous *VS* Synchronize

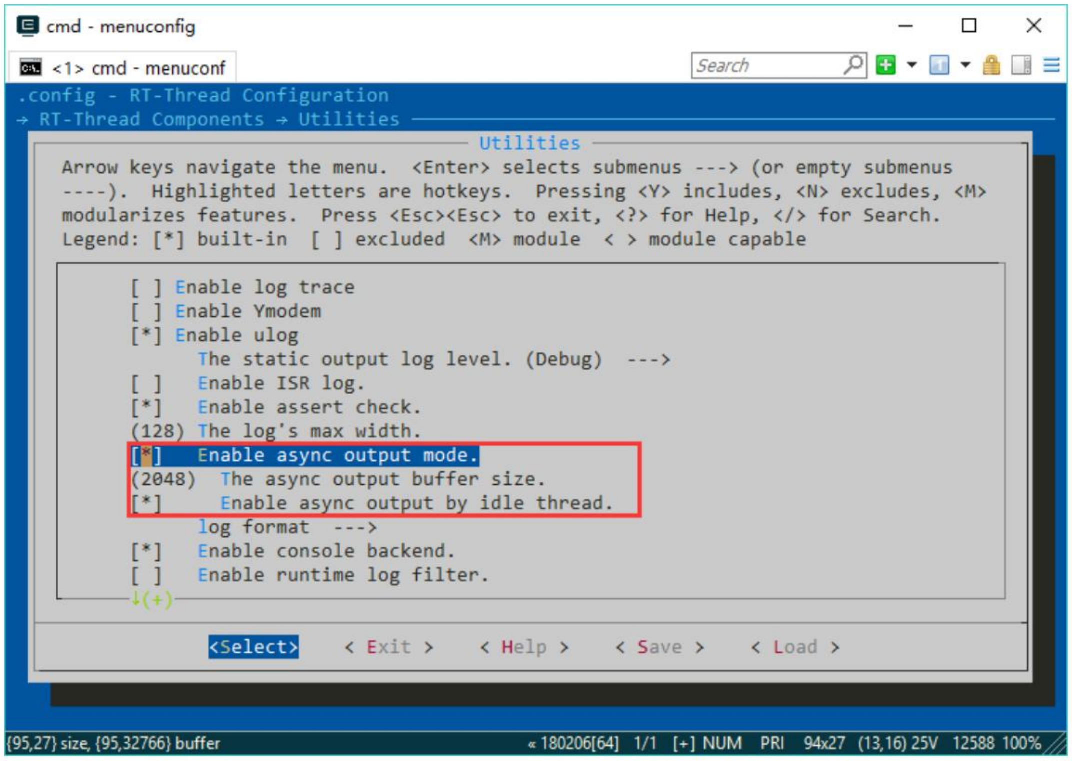
Let's take a look at the advantages and disadvantages of asynchronous mode

- advantage:
  - First, the log output will not block the current thread, and some backends have low output rates, 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 also be reduced.
  - In synchronous mode, interrupt logs can only be output to the console backend, while in asynchronous mode, interrupt logs can be output
    To all backends.
- Disadvantages: First, the asynchronous mode requires a log buffer. Second, the output of asynchronous logs also requires a dedicated thread to complete, such as an idle thread or a user-defined thread, which is slightly complicated to use. Overall, the asynchronous mode uses more resources than the synchronous mode.

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, the effect of turning it on is as follows:



```
.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. ÿAsynchronous buffer size, default is 2048 • Enable async output by idle thread. ÿWhether

to let the idle thread as the asynchronous log output thread, so that when the system is idle, the log output can be automatically completed. 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

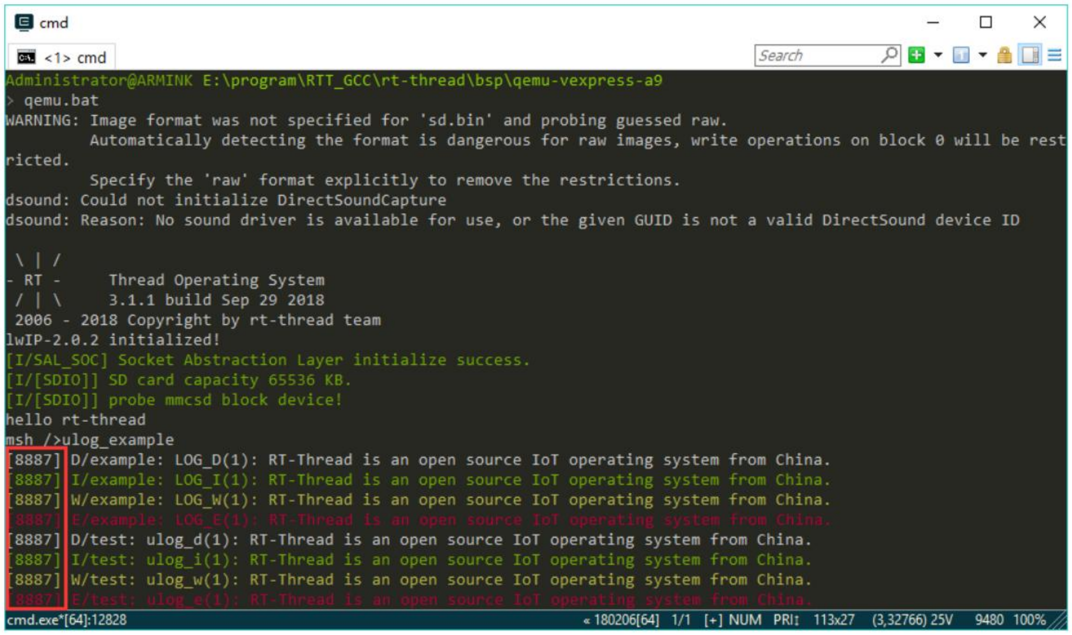
configuration • Copy rt-thread\examples\ulog_example.c to rt-thread\bsp\qemu-vexpress-a9\

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.



```
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 restricted.
         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.
```

Figure 4: *ulog* Asynchronous routines

If you observe carefully, you can find that after the asynchronous mode is turned on, the time information of these logs that are very close in the code is almost the same. However, in the synchronous mode, the logs are output using user threads. Since log output takes a certain amount of time, there will be a certain interval between each log. This also fully demonstrates that the output efficiency of asynchronous logs is very high and it hardly takes up the caller's time.

3.3 Log Filter (Dynamic Filter)

In the basic chapter of the ulog application note, some static filtering functions of logs are introduced. Static filtering has its advantages, such as saving resources, but in many cases, users need to dynamically adjust the filtering method of logs when the software is running, which can use the dynamic filter function of ulog. To use the dynamic filter function, you need to turn on the Enable runtime log filter option in menuconfig, which is turned off by default.

Ulog supports the following four dynamic filtering methods, and each has a corresponding API function and Finsh/MSH command, 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 module referred to here is introduced in the basic part of the application note, representing a type of log code with the same tag attribute. Sometimes you need to dynamically modify the log output level of a module at runtime, for example: to turn off the log of the wifi module, you can set the log output level of the wifi module to LOG_FILTER_LVL_SILENT .

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 the global filtering level is set 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, the tag filter is canceled

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

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

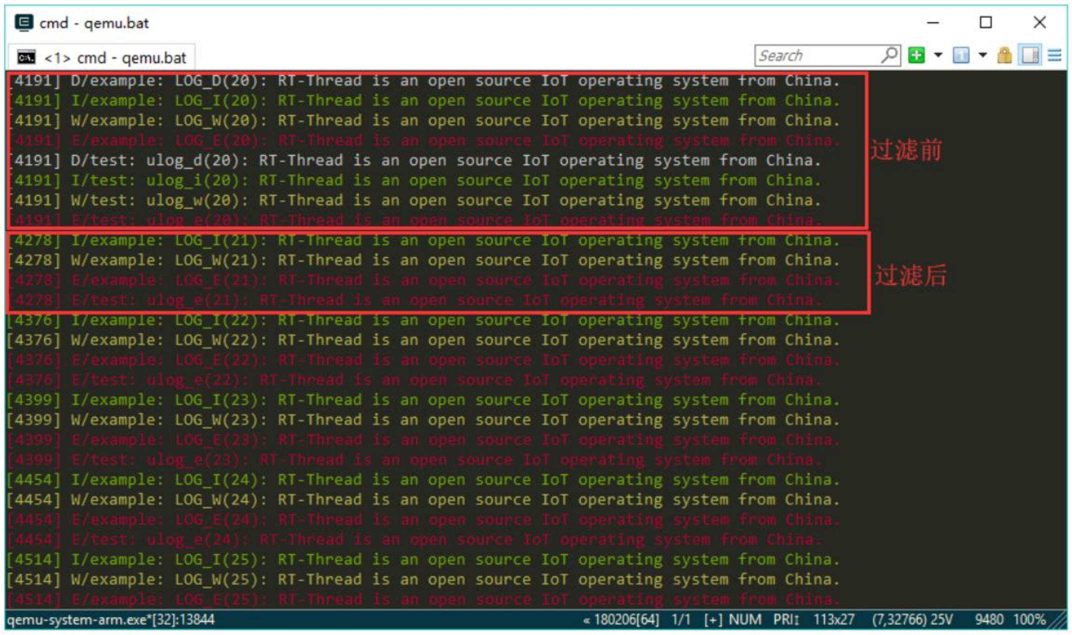
3.3.5. Run the routine

Still executed in qemu BSP, first enable dynamic filtering in menuconfig, then save the configuration and compile, Run the example program. After the log is output about **20** times, the following filter 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);
}
...
```

At this time, the global filtering level is set to INFO, so logs lower than INFO level can no longer be seen. At the same time, the log output level of the test tag is set to ERROR, and the logs lower than ERROR in the test tag are also stopped from being output. In each log, there is a count value of the current log output times, and the comparison effect is as follows:


cmd - qemu.bat
<1> cmd - qemu.bat
Search
过滤前
过滤后
qemu-system-arm.exe*[32]:13844
« 180206[64] 1/1 [+] NUM PRI↕ 113x27 (7,32766) 25V 9480 100%

```
[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.
```

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

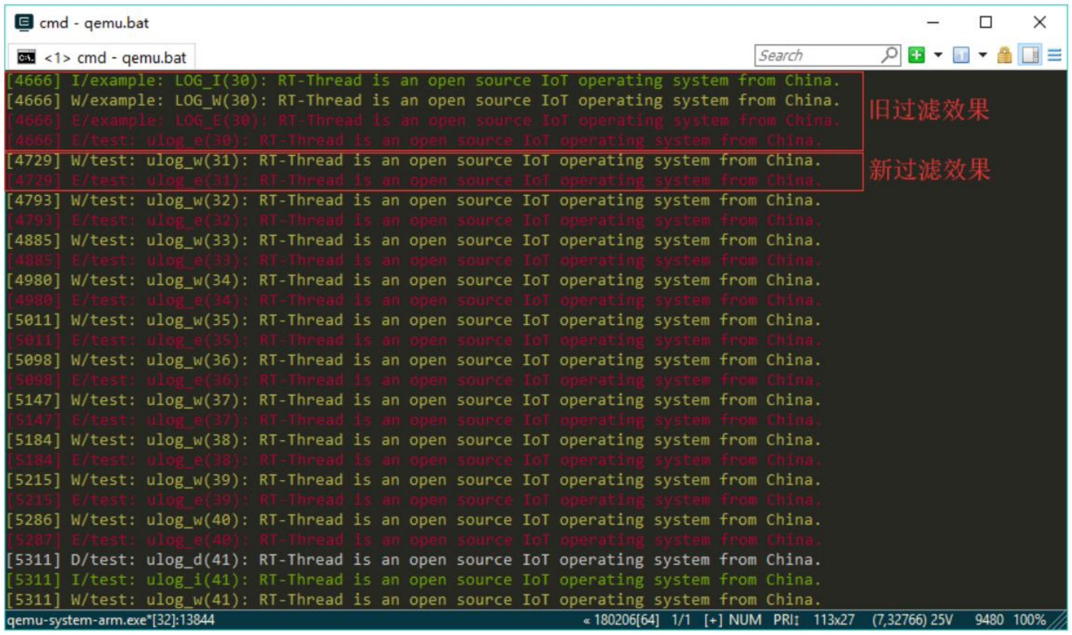
After the log is output about **30** times, the corresponding filtering code in ulog_example.c will be executed:

```
...
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);
}
...
```

At this time , a filter for the example module is added , and all logs of this module are stopped from being output, so the module logs will not be visible next. At the same time, the log output level of the test tag is reduced to WARING, so that only the WARING and ERROR level logs of the test tag can be seen. The effect is as follows:



```
[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%
```

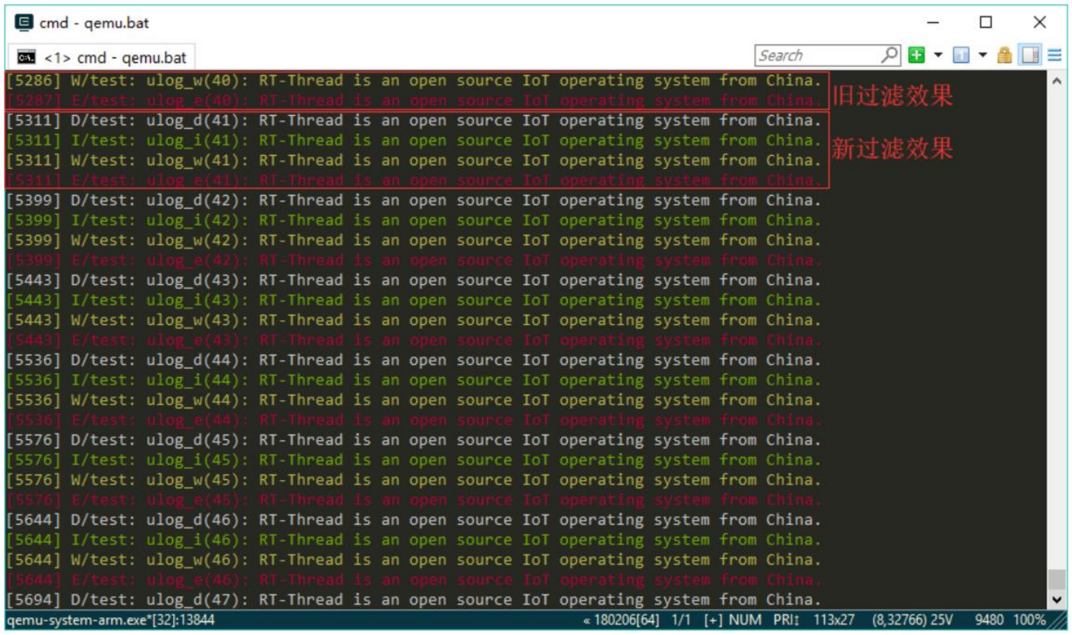
Figure **6:** *ulog* Filter routines *30*

After the log is output about **40** times, the corresponding filtering 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 time, the log output level of the test module is adjusted to LOG_FILTER_LVL_ALL , that is, the logs of any level of 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 resume output. The effect is as follows:


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 Use when the system is abnormal

Since ulog's asynchronous mode has a cache mechanism, the registered backend may also have a cache. If the system encounters hardfault, assertion or other errors, but there are logs in the cache that have not been output, this may cause log loss, and it will be 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. Assertions

RT-Thread's assertion supports assertion callbacks (hooks). 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);
}
```

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 the 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 the syslog mode, no matter which log output API is used, the entire ulog log output format will use 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

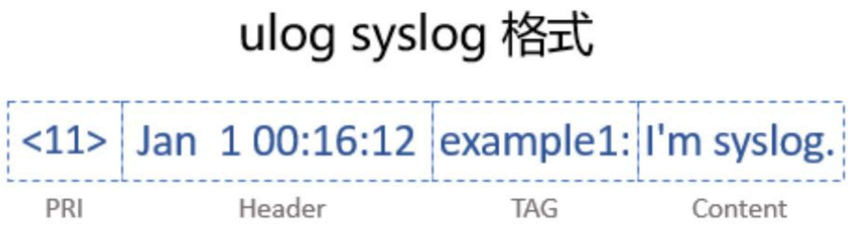


Figure 8: *ulog syslog* Format

As shown in the figure above, the ulog syslog log format is divided into the following 4 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 Facility by 8 and then adding Severity. Facility and Severity are passed in through the input parameters of 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 through the openlog function input parameter. If not specified, rtt will be used As the default label;
- **Content :** The specific content of the log.

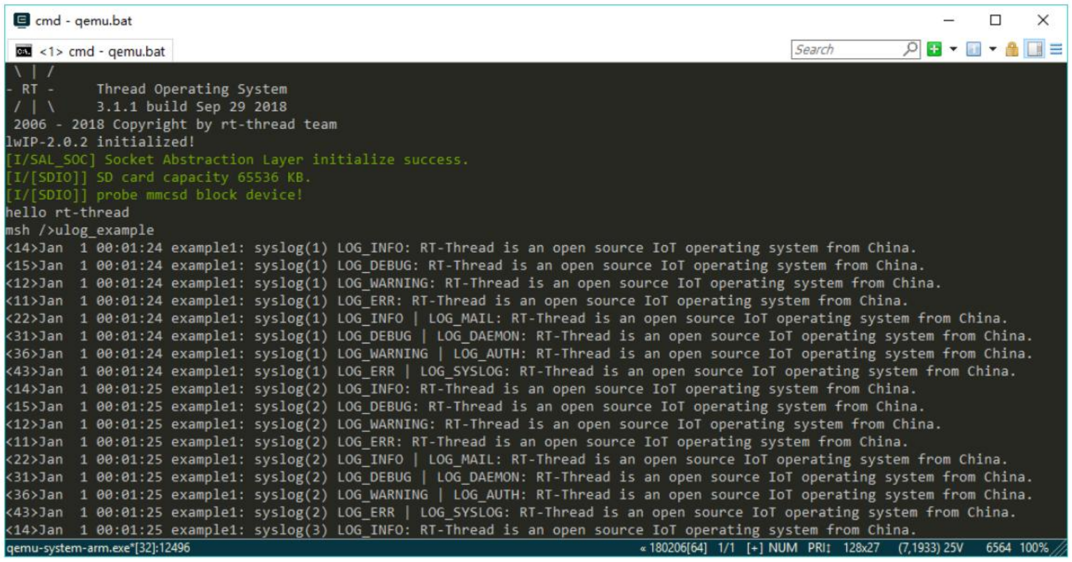
3.5.3. Usage

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 called automatically the first time syslog is called.

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



```
 \ | /
- 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

With log tools, if they are used improperly, they may be abused and the log information may not highlight the key points. Here, we will focus on sharing some tips on using log components to make log information more intuitive. The main points of concern 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 you first start using the log library, you will often encounter problems such as being unable to distinguish between warning and error logs, and between information and debug logs, which leads to inappropriate log level selection. Some important logs may not be visible, and unimportant logs are everywhere. Therefore, before use, be sure to carefully read the log level section of "RT-Thread ulog Log Component Application Notes - Basics", which has clear standards for each level.

3.6.3. Avoid redundant logging

In some cases, the code may be called repeatedly or executed in a loop, and the same or similar logs may be output multiple times. Such logs will not only take up a lot of system resources, but also affect the developers' ability to locate the problem. Therefore, when encountering such a situation, it is recommended to add special processing for repetitive logs, such as: let the upper layer output some business-related logs, and the lower layer only returns the specific result status; can the same logs at the same time point be deduplicated, and only output once when the error status does not change, etc.

3.6.4. Enable more log formats

The default log format of ulog does not have timestamp and thread information enabled. These two log information are very useful on RTOS. They can help developers intuitively understand the running time and time difference of each log, and can also clearly see which thread is executing the current code. So if conditions permit, it is recommended to enable them.

3.6.5. Turn off unimportant logs

Ulog provides multiple dimensions of log switch 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 currently debugged module. For more information about the log filtering function, 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 this prompt is encountered, it indicates that the buffer overflows in asynchronous mode, which will cause some logs to be lost. Increasing the ULOG_ASYNC_OUTPUT_BUF_SIZE option can solve this problem.

- 2. Compilation 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 the idle thread as the output thread, the stack size of the idle thread needs to be increased, which also depends on the specific backend device. For example, for the console backend, the idle thread must be at least 384 bytes.

5 References

5.1 All relevant **APIs** in this article

5.1.1. **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

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 registering, 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 receiving data length
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. Filter logs by module/tag level

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

This API can be used to filter logs according to the level of the tag, 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 INFO level of wifi.mgnt module*/
```

```
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. Filter 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, see the table above, also see ulog_def.h

5.1.7. Filter logs by tag (global)

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

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

Output.

parameter	describe
tag	Set filter tags

5.1.8. Filter 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