EC2X&AG35-QuecOpen

Log System User Guide

LTE Module Series

Rev. EC2X&AG35-QuecOpen_Log_System_User_Guide_V1.0

Date: 2018-07-17

Status: Preliminary

Our aim is to provide customers with timely and comprehensive service. For any assistance, please contact our company headquarters:

Quectel Wireless Solutions Co., Ltd.
7th Floor, Hongye Building, No.1801 Hongmei Road, Xuhui District, Shanghai 200233, China
Tel: +86 21 5108 6236
Email: info@quectel.com

Or our local office. For more information, please visit:
http://quectel.com/support/sales.htm

For technical support, or to report documentation errors, please visit:
http://quectel.com/support/technical.htm
Or email to: support@quectel.com

GENERAL NOTES

QUECTEL OFFERS THE INFORMATION AS A SERVICE TO ITS CUSTOMERS. THE INFORMATION PROVIDED IS BASED UPON CUSTOMERS' REQUIREMENTS. QUECTEL MAKES EVERY EFFORT TO ENSURE THE QUALITY OF THE INFORMATION IT MAKES AVAILABLE. QUECTEL DOES NOT MAKE ANY WARRANTY AS TO THE INFORMATION CONTAINED HEREIN, AND DOES NOT ACCEPT ANY LIABILITY FOR ANY INJURY, LOSS OR DAMAGE OF ANY KIND INCURRED BY USE OF OR RELIANCE UPON THE INFORMATION. ALL INFORMATION SUPPLIED HEREIN IS SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.

COPYRIGHT

About the Document

History

Revision	Date	Author	Description
1.0	2018-07-17	Tyler KUANG	Initial

Contents

1 Introduction

Log system in QuecOpen, take LOGCAT solution of Android. LOGCAT includes 4 ring buffer (MAIN, RADIO, EVENTS, SYSTEM), 6 log level (VERBOSE, DEBUG, INFO, WARN, ERROR, FATAL).

In QuecOpen, the buffer is used as follows:
MAIN: Used for client application process.
RADIO: Deprecated.
EVENTS: Record system events.
SYSTEM: Record the log of critical system service process.

Use logcat command to check log and according to TAG and log level to filter. For specific usage please refer to https://developer.android.com/studio/command-line/logcat.

2 The Using and Checking of Log Interface

2.1. The Using of Log

The log writing interface takes liblog encapsulated by Android, or qlsyslog encapsulated by QuecOpen, and later chapter will mainly introduces the usage of qlsyslog.

Header file: qlsyslog/ql_sys_log.h
Library: libql_sys_log.so

The log interface definition is located at header file qlsyslog/ql_sys_log.h. The usage of log interface please refer to below code, and in Makefile, please add -lql_sys_log to link option.

```
#include"qlsyslog/ql_sys_log.h"

#define LOG_TAG "fortest"

int main(int argc, char *argv)
{
    QLOGV(LOG_TAG, "I am QL_SYS_LOG_VERBOSE");
    QLOGD(LOG_TAG, "I am QL_SYS_LOG_DEBUG");
    QLOGI(LOG_TAG, "I am QL_SYS_LOG_INFO");
    QLOGW(LOG_TAG, "I am QL_SYS_LOG_WARN");
    QLOGE(LOG_TAG, "I am QL_SYS_LOG_ERROR");
    QLOGF(LOG_TAG, "I am QL_SYS_LOG_FATAL");

    return 0;
}
```

2.2. The Checking of Log

Please check log through logcat tool, and for the user guide of logcat tool please refer to https://developer.android.com/studio/command-line/logcat

Take code in Chapter 2.1 as example to introduce the use mode of logcat.

- Filter log via TAG

```
/data # logcat -s fortest
--------- beginning of system
--------- beginning of main
07-17 12:24:11.472  2039  2039 V fortest : I am QL_SYS_LOG_VERBOSE
07-17 12:24:11.472  2039  2039 D fortest : I am QL_SYS_LOG_DEBUG
07-17 12:24:11.472  2039  2039 I fortest : I am QL_SYS_LOG_INFO
07-17 12:24:11.472  2039  2039 W fortest : I am QL_SYS_LOG_WARN
07-17 12:24:11.472  2039  2039 E fortest : I am QL_SYS_LOG_ERROR
07-17 12:24:11.472  2039  2039 F fortest : I am QL_SYS_LOG_FATAL
```

- Filter log via log level

```
/data # logcat -s fortest:w
--------- beginning of system
--------- beginning of main
07-17 12:24:11.472  2039  2039 W fortest : I am QL_SYS_LOG_WARN
07-17 12:24:11.472  2039  2039 E fortest : I am QL_SYS_LOG_ERROR
07-17 12:24:11.472  2039  2039 F fortest : I am QL_SYS_LOG_FATAL
```

- Filter log via buffer

```
/data # logcat -b main
07-17 12:24:11.472  2039  2039 V fortest : I am QL_SYS_LOG_VERBOSE
07-17 12:24:11.472  2039  2039 D fortest : I am QL_SYS_LOG_DEBUG
07-17 12:24:11.472  2039  2039 I fortest : I am QL_SYS_LOG_INFO
07-17 12:24:11.472  2039  2039 W fortest : I am QL_SYS_LOG_WARN
07-17 12:24:11.472  2039  2039 E fortest : I am QL_SYS_LOG_ERROR
07-17 12:24:11.472  2039  2039 F fortest : I am QL_SYS_LOG_FATAL
```

3 Log Persistence Configuration

Service process qlllog will save kernel log and application log to configuration file according to filtering rules, and configuration file is located at /data/qllog.json, format is json, the main configuration options are shown as follow.

Table 1: Main Configuration Options

ITEM	Option	Description
log_file	Required	The path which log file saved in, please don't save log in critical system partition.
rotate_file_size	Required	The limitation of single log file size, in KB.
rotate_file_count	Required	Maximum number of log files.
log_format	Optional	Log file output format, options are default, CSV
kernel_priority	Optional	Kernel log level, i.e. printk level, options are m(emerg), a(alert), c(crit), e(err), w(warn), n(notice), i(info), d(debug), *(all level)
buffer_list.{i}.name	Required	Buffer Name, options are main, system, events.
buffer_list.{i}. filter_list.{i}.tag	Optional	TAG of the log which needed saved, if not write, default all TAG.
buffer_list.{i}.filter_list. {i}.priority	Optional	Level of the log which needed saved, options are v(VERBOSE), d(DEBUG), i(INFO), w(WARN), e(ERROR), f(FATAL), *(all level)

4 How Does the Application Record Exceptions

4.1. How to Monitor Application Exceptions

The application record exceptions by monitoring signal, example codes are shown as bellow. In the exception handler, ql_sys_log_signal(qlsyslog/ql_sys_log.h) will log as much exception information as possible.

```
#include <signal.h>
#include <stdlib.h>
#include"qlsyslog/ql_sys_log.h"

#define LOG_TAG "fortest"

static void handle_signal (int sig_num, siginfo_t *info, void *ptr)
{
    ql_sys_log_signal(QL_SYS_LOG_ID_MAIN, QL_SYS_LOG_FATAL, LOG_TAG, sig_num, info, ptr);
    exit(-1);
}

int main(int argc, char *argv)
{
    struct sigaction sa = {0};
    sa.sa_sigaction = handle_signal;
    sa.sa_flags = SA_SIGINFO;
    sigaction (SIGTERM, &sa, NULL);
    sigaction (SIGSEGV, &sa, NULL);
    sigaction (SIGABRT, &sa, NULL);
    sigaction (SIGINT, &sa, NULL);
    sigaction (SIGBUS, &sa, NULL);

    QLOGI(LOG_TAG, "bootup");

    /* other code */

    return 0;
```

```
}
```

4.2. How to get exception call stack

When there is an exceptions in application, users often want to get a detailed application call stack to locate the exception code. Ql_sys_log_signal can backtrack function call stack, but there are some limitations.

Method: In compilation option of application, delete -O1,-O2,-O3, -fomit-frame-pointer optimization option and add -fasynchronous-unwind-tables –rdynamic compilation option.

Limitation: Some libraries add optimization options when compiling. If an exception occurs in a library function, it may not be possible to trace back to the detailed call stack.

5 Attention

During the running of the system, a large number of logs may be generated, and log files are frequently written. If the log file is stored in the flash, the flash life will be shortened (the flash technology parameter has the maximum number of writes). Therefore, it is recommended to save the log in the flash during the development for debugging, turn off the log save function (do not startup qllog) or output the log file to the temporary file system.