Long River
Vision Systems

USER'S Guide

For Embedded C Logging

Prepared by: P.Shi
Created: 04/23/2017
Last Updated: 04/24/2016

Rev: 1

Revision history

Release No.	Date	Comment
Rev. 1	4/23/2017	Initial draft

TABLE OF CONTENTS

1. INTRODUCTION

Embedded C Logging is a special solution for embedded Linux platform where system log or console output is not available.

The Embedded C logging was developed with stand C language (c89) and standard c libraries and can be compiled by cross compiler such as gcc based cross compiles without any code changes.

The Embedded C logging was also provided as a compiled library on Ubuntu Linux platform for free evaluation. However for embedded development platform such as ARM based embedded Linux kernel, the easiest way is to incorporate the source code into your project (need source code license) . The code size is lightweight (less than 2K lines) and can be easily customized to meet your requirement.

1.1. Product Overview

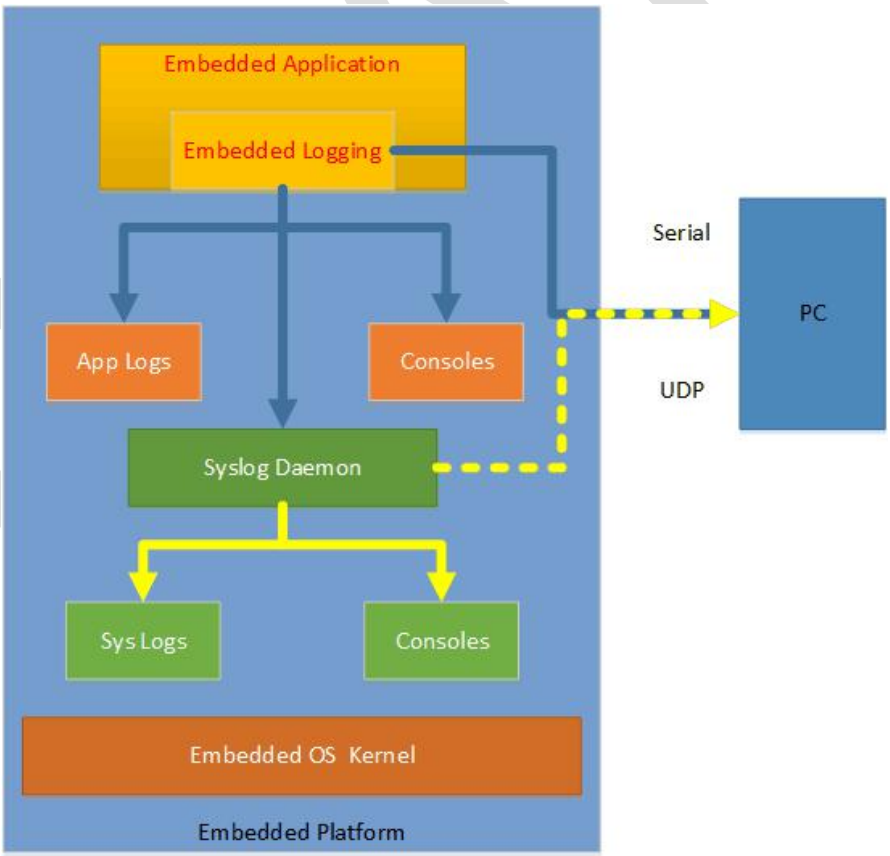

Figure 1. Embedded C logging architecture

The target of Embedded C logging is ucLinux and embedded Linux kernel platform and has following features:

- Easy integration: The user just need include 1 header file (log_api.h) and link its library) into the project. if there is no compatible library found, the user need incorporate all source code (3 C files) into the project.
- All log configurations are stored in one file and initially loaded during logging initialization.
- All log configurations can be overridden/adjusted at running time by the same names environment variables.
- The logging information based on logging level(such as warning, error and debug messages) can be logged locally or remotely , one or multiple locations.
- Embedded C logging does not rely on system syslog but still can redirect syslog message (RFC3164 or RFC5424) to remote server. Working with the remote capture tool that runs on PC(Windows or Linux) , it can send the logging information over UDP socket and display colorful messages.
- Embedded C logging provides simple logging rotations by checking logging files and number of files, which is especially suitable for flash based embedded system.
- Embedded C logging API is thread safe. You can any log API in different threads without worrying message being blocked or truncated.

1.2. About This Manual

The level of coverage provided in this manual requires that you have basic c language programming skill and understand UNIX/LINUX system logging mechanism. The potential reader is embedded software engineer who wants to find lightweight ,time saving embedded debugging solution , or the beginners that just get out IDE development environment and start exploring embedded world.

Disclaimer

The LRVS does not assume any liability arising out of the application or use of any products, or software described herein. This documentation is subject to change without notice. Please visit http://www.lrvs.net for the latest version.

2. GETTING STARTED

2.1. Get Embedded C logging source code

Embedded C logging library only contains following 6 files:

- log_api.h(header file) - main API interface
- log_api.c(embedded logging main source file) -main Logging implementation
- log_serial.h - Serial communication output header file
- log_serial.c - Serial output implementation file
- log_udp.h - UDP output header file.
- log_udp.c - UDP output function body file

2.2. Integrate Embedded C logging

Embedded C logging can be integrated into your project in two ways:

- Get embedded C logging as a dynamic or static library, include “log_api.h” and link the library into your project.See the example in the appendix A.
- Add all embedded C logging source code (need license) into your project and compiled by your cross compiler of target machine. you can fully take advantage all logging APIs.

2.3. Start using Embedded C Logging APIs

Please include “log_api.h” in every file prior to using any APIs,

1. Embedded C logging need be initialized first with

```
int LogApi_Init( const char *config_file);
```

Where **config_file** is logging configuration file name , which is relative to current running programming. The configuration file defines all logging behaviors, such as log level, output destination, log format etc. Please refer to Appendix C for a sample of configuration file.

2. Add any logging/debugging messages in your source code with APIs/macros defined in log_api.h(see detail description in section 4).

3. The Embedded C logging can be reinitialized with new configuration file with

```
int LogApi_Reinit( const char *config_file );
```

If the program is build and running, the user can update logging parameters by adding and modifying their environment variables followed with sending “USR1” or “USR2” signal to the running process which will make the new parameters take effect immediately.

4. Exit embedded C logging with

```
int LogApi_Term(void);
```

3. LOGGING PARAMTERS

This section discusses various logging parameters that can be used in configuration file and environment variables.

syslog_enabled: - [0- disabled; 1 - enabled] Enable or disable syslog:

Note: this parameter does not control syslog daemon and only controls whether message is output to syslog daemon or not . If you want to log messages via the syslog daemon, you not only need set syslog_enable =1, you should also ensure that your syslogd daemon is running. syslog's configuration file is usually in /etc/syslog.conf and the log files by syslog are usually at /var/log. You may refer to syslog document on how to setup or control syslog, which is not covered in this document.

If your embedded system kernel does not have in-house syslog daemon ,the Embeded C logging can implement similar functions that syslog daemon offers, such as send syslog (RFC3164 or RFC5424 standard) messages to remote syslog server etc.

facility: [0-7] syslog facility for remote logging.This parameter is used to calculate syslog priority, e.g.

if facility=0 (LOG_LOCAL0) and log_level =7

priority of syslog = (16<<3) + LOGAPP_DEBUG(7) = 135

And log output over application UDP:

<135>Apr 22 17:56:18 ThinkPad-X40 myproc[2541]: L7: DM6467: deug output

Facility	Syslog facility	Priority
0	LOG_LOCAL0	16 << 3
1	LOG_LOCAL1	17 << 3
2	LOG_LOCAL2	18 << 3
3	LOG_LOCAL3	19 << 3
4	LOG_LOCAL4	20 << 3
5	LOG_LOCAL5	21 << 3
6	LOG_LOCAL6	22 << 3
7	LOG_LOCAL7	22 << 3

Table 1. Syslog facility definitions

destination – sets destination for log messages. (bitwise)
0 – stdin (unused)
1 – stdout (console) – default – bit 0
2 - stderr (standard error console) -bit 1
3 - (unused)
4 : - File
5 or – console and file
7 – unused.
8 – Serial port: - bit 3

9-10 – console and serial port
12 – File and serial port
13 – Console,File and serial port
16 - UDP : bit 4:

For example,if the user wants to output same message to stderr, File and Serial port, destination will be 2 + 4 + 8 = 14; i.e.,
destination = 14

Note: logging destination for syslog is not controlled by Embedded C logging. If you want to redirect log messages to specific destination such as file or remote host, you need to locate and edit the **syslog.conf** file on your system, which is out of scope of this document.

Warning: Enabling multiple destinations esp. serial output may slow down your running process.

log_level: [-5 ~7]- logging running level

Only log message whose level is no more than this level will be output. The log message whose level is larger than this level will be suppressed. Among them , log level from -5 (SYS_LOG_PNC) to -1 (SYS_LOG_INFO) is for syslog, which also means all application levels are disabled.

Log level	Local syslog severity(syslog)	Remote syslog severity
-5 (SYS_LOG_PNC)	LOG_ERR	
-4(SYS_LOG_FAT)	LOG_ERR	
-3(SYS_LOG_ERR)	LOG_ERR	
-2(SYS_LOG_WAR)	LOG_WARNNING	
-1(SYS_LOG_INFO)	LOG_INFO	
0(LOGAPP_EMERG)	LOG_INFO	LOG_EMERG (0)
1(LOGAPP_ALERT)	LOG_INFO	LOG_ALERT (1)
2(LOGAPP_CRIT)	LOG_INFO	LOG_CRIT (2)
3(LOGAPP_ERROR)	LOG_INFO	LOG_ERROR (3)
4(LOGAPP_WARN)	LOG_INFO	LOG_WARN (4)
5(LOGAPP_NOTICE)	LOG_INFO	LOG_NOTICE (5)
6(LOGAPP_INFO)	LOG_INFO	LOG_INFO (6)
7(LOGAPP_DEBUG)	LOG_INFO	LOG_TRACE (7)

Table 2. Log level definitions

log_filename : debugging log file name (relative to running folder),

e.g., assuming the running process is in /usr/bin, the following setting will redirect log files to /tmp

log_filename = ../../tmp/myproc.log

number_of_files : number of files to rotate.e.g.,

number_of_files = 5

max_logfile_size: maximum size of one logfile (in KB)

max_logfile_size = 512

Simple Log rotation mechanism - Taking above settings as an example:

myproc.log (new) -> myproc.log1 -> myproc.log2->myproc.log3->myproc.log4-> myproc.log5 (oldest) , myproc.log6 (deleted).

serial_com : serial communication port for serial destination , such as

Serial_com = /dev/ttyUSB0

udp_host: UDP destination - host IP

udp_port: UDP destination - port number e.g.

udp_host = 192.168.1.17

udp_port = 12345

If UDP destination is enable, the debugging message will be direct to udp_host:up_port.

log_format: [0-3] Log format mode:

Log format	description
0	Time stamp mode:MM/DD/YY %H:%M:%S.mmm, such as Aug 24 05:14:15,e.g. 04/13/17 15:55:46.971 myproc[1518].L7-main.c/53 [DM6467: debug log]
1	Runtime mode: Elapse time (sec: msec) from program running,.e.g 03.375 myproc[1518].DB7-main.c/53: DM6467: debug log
2	syslog compatible mode:RFC3164 <priority>Mm DD HH:MM:SS host tag[pid]: msg e.g: <134>Apr 22 17:56:18 ThinkPad-X40 myproc[2541]: L6: DM6467: Info output

3	syslog mode: RFC5424 <priority>version YYYY-MM-DDTHH:MM:SS.MMMZ host proc pid msgid structured_data msg, e.g. <131>1 2017-04-22T18:02:31.824Z ThinkPad-X40 myproc 2580 L3 - : DM6467: Error output

Table 3. Log format definitions

Note: When you do remote logging over UDP interface with log_format=0 or 1 , messages are sent with encoded colors so that they can be displayed different colors with the remote capture tool. It still be readable with third party UDP trace tool.

If local syslog daemon is disable or not available, the user still can send debugging messages to remote syslog server over UDP. For this case, please choose log_format =2 or 3, RFC3164 or RFC5424;

4. API USAGES

4.1. System level Log API

- LogSys_Info(params...)
- LogSys_Warning(params...)
- LogSys_Error(params...)
- LogSys_Fatal(params...)
- LogSys_Panic(params...)
- LogSys_Mark()
- LogSys_Log()

Description

These macros are used to leave critical or system level logs. Usually even though application level is disabled (log_level <)) , these APIs still output messages to syslog daemon.

Example

LogSys_Info(“ Leave syslog message”);

Output:

/var/log/syslog

Apr 25 21:49:49 ThinkPad-X40 myproc.log[2576]: [INFO] Leave syslog message

4.2. Application level Log API

- LogApp_Emerg(params...)
- LogApp_Alert(params...)
- LogApp_Crit(params...)
- LogApp_Error(params...)
- LogApp_Warn(params...)
- LogApp_Notice(params...)
- LogApp_Info(params...)
- LogApp_Debug(params...)

Description

These macros are used to leave application logs.

Note: Only macros whose debugging levels are less than predefined log_level output messages to preset destination. The user can run-time control message throughput by adjusting ‘log level’ environment variables. Such as, setting log_level < 0 will disable all Application logs.

Example

LogApp_Debug(“ Trace log”);

Output only when log_level =7

Apr 25 21:49:49 peitao-ThinkPad-X40 myproc.log[2576]: [L7] DM6467: Trace log

4.3. Hex Dump API

- LogApp_HexDump_Emerg(title,data,size)
- LogApp_HexDump_Alert(title,data,size)
- LogApp_HexDump_Crit(title,data,size)
- LogApp_HexDump_Error(title,data,size)
- LogApp_HexDump_Warn(title,data,size)
- LogApp_HexDump_Notice(title,data,size)
- LogApp_HexDump_Info(title,data,size)
- LogApp_HexDump_Debug(title,data,size)

Description

These specific hex-dump macros(application levels) contain 3 parameters: data name , the pointer to the data, and data length to be printed._

Example

```
char* myData=
"1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSKUVWXYZ!@#$%^&*()";
```

LogApp_HexDump_Debug("myData",myData, strlen(myData));

Output when log_level = 7

```
4.019 myproc[3014].L7-main.c/59: myData - Length=72
4.019 myproc[3014].L7-main.c/59: 31 32 33 34 35 36 37 38 39 30 61 62 63 64 65 66|1234567890abcdef
4.019 myproc[3014].L7-main.c/59: 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76|ghijklmnopqrstuv
4.019 myproc[3014].L7-main.c/59: 77 78 79 7A 41 42 43 44 45 46 47 48 49 4A 4B 4C|wxyzABCDEFGHIJKL
4.019 myproc[3014].L7-main.c/59: 4D 4E 4F 50 51 52 53 4B 55 56 57 58 59 5A 21 40|MNOPQRSKUVWXYZ!@
4.019 myproc[3014].L7-main.c/59: 23 24 25 5E 26 2A 28 29                         |#$%^&*()
```

4.4. Colorful printf APIs

- PRINT_RED(s...)
- PRINT_GREEN(s...)
- PRINT_YELLOW(s...)
- PRINT_BLUE(s...)
- PRINT_MAGENTA(s...)
- PRINT_CYAN(s...)

These macro API output messages to “stdout”with different colors. Prior to using them, please make sure your console can support color display.

Example

PRINT_GREEN(“DM6467 --- printing warning!”);

Output:

4.5. Printf()-log level control API

- printf_error(const char *fmt, ...);
- printf_warn(const char *fmt, ...);
- printf_debug(const char *fmt, ...);

Descriptions: These macro APIs output messages to “stdout” with default color when its log level is less than predefined log level. These APIs are also thread safe.

Example

printf_error("DM6467 --- printing error!");
printf_warn("DM6467 --- printing warning!");
printf_debug("DM6467 --- printing debug!");

Output:

5. REMOTE LOGGING/TRACING

Embedded c logging can send debugging messages to remote host over UDP packets. To be compatible with syslog sever, it can also send RFC3124 or RFC5424 compliant messages to remote site.

5.1. UDP Message Format

When sending log message to remote host with log_format=0,1,
The UDP packet is prefixed one byte color code:

UDP message will be encoded with the following color code:

Value	Color
0xE0	Black
0xE1	Blue
0xE2	Green
0xE3	Cyan
0xE4	Red
0xE5	Magenta
0xE6	Yellow
0xE7	White
0xE8	Gray
0xE9	Light blue
0xEA	Light green
0xEB	Light cyan
0xEC	Light red
0xED	Light magenta
0xEE	Light yellow
0xEF	Light white

Table 4: UDP message color code

Embedded C log predefined different color for different log level. To change different colors, the user need access log_udp.h APIs

Level	Color code
LOGAPP_EMERG	Gray
LOGAPP_ALERT	Light blue
LOGAPP_CRIT	Light green
LOGAPP_ERROR	Light red

LOGAPP_WARN	Light yellow
LOGAPP_NOTICE	Light magenta
LOGAPP_INFO	Light white
LOGAPP_TRACE	Light green

Example:

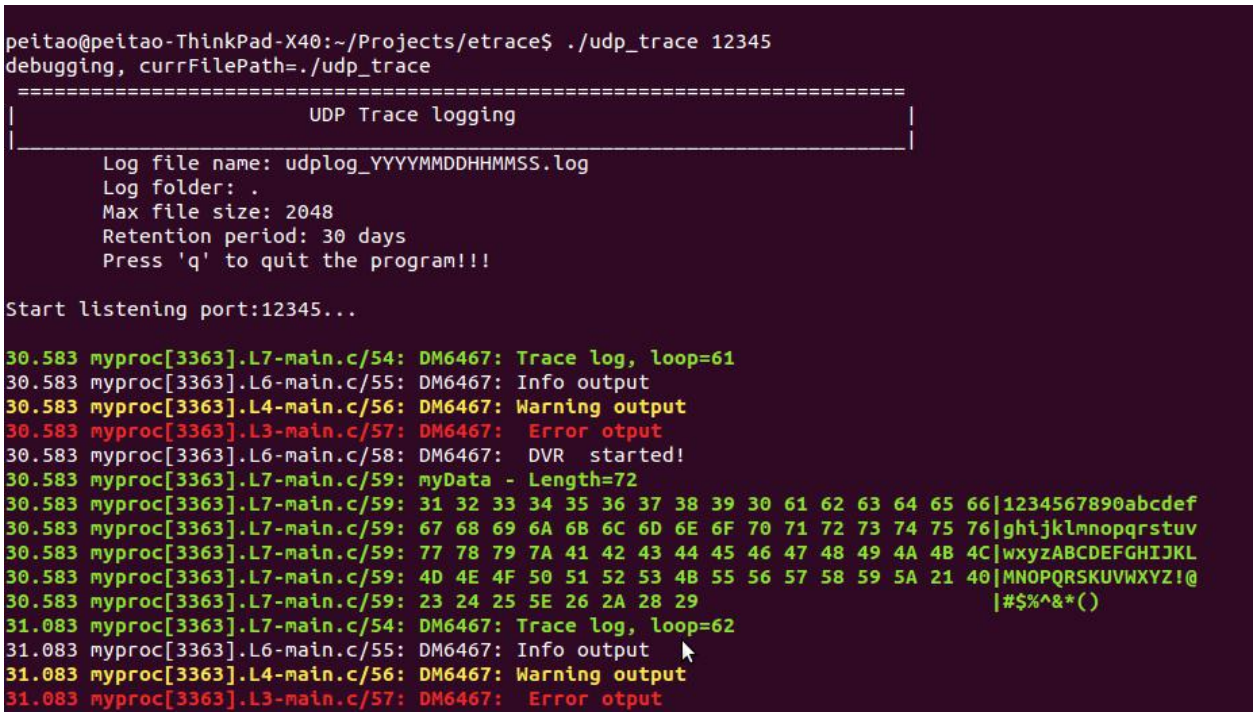

Figure 2. Etrace for logging -Linux version

However,if the user selects the compatible syslog (RFC3124 or RFC5424) format output,there is no color displayed

```
peitao@peitao-ThinkPad-X40:~/Projects/etrace$ ./udp_trace 12345
debugging, currFilePath=./udp_trace
 ========================================================================
|                        UDP Trace logging                               |
|________________________________________________________________________|
        Log file name: udplog_YYYYMMDDHHMMSS.log
        Log folder: .
        Max file size: 2048
        Retention period: 30 days
        Press 'q' to quit the program!!!

Start listening port:12345...

<127>1 2017-04-25T23:32:43.056Z peitao-ThinkPad-X40 myproc 3431 INFO - : Log Level = 7, Log file = myprog.
log
<127>1 2017-04-25T23:32:43.058Z peitao-ThinkPad-X40 myproc 3431 INFO - : Leave syslog message
<135>1 2017-04-25T23:32:43.059Z peitao-ThinkPad-X40 myproc 3431 L7 - : DM6467: Trace log, loop=0
<134>1 2017-04-25T23:32:43.061Z peitao-ThinkPad-X40 myproc 3431 L6 - : DM6467: Info output
<132>1 2017-04-25T23:32:43.061Z peitao-ThinkPad-X40 myproc 3431 L4 - : DM6467: Warning output
<131>1 2017-04-25T23:32:43.061Z peitao-ThinkPad-X40 myproc 3431 L3 - : DM6467:  Error otput
<134>1 2017-04-25T23:32:43.061Z peitao-ThinkPad-X40 myproc 3431 L6 - : DM6467:  DVR  started!
<135>1 2017-04-25T23:32:43.061Z peitao-ThinkPad-X40 myproc 3431 L7 - : myData - Length=72
<135>1 2017-04-25T23:32:43.061Z peitao-ThinkPad-X40 myproc 3431 L7 - : 31 32 33 34 35 36 37 38 39 30 61 62
 63 64 65 66|1234567890abcdef
<135>1 2017-04-25T23:32:43.061Z peitao-ThinkPad-X40 myproc 3431 L7 - : 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72
 73 74 75 76|ghijklmnopqrstuv
<135>1 2017-04-25T23:32:43.061Z peitao-ThinkPad-X40 myproc 3431 L7 - : 77 78 79 7A 41 42 43 44 45 46 47 48
 49 4A 4B 4C|wxyzABCDEFGHIJKL
<135>1 2017-04-25T23:32:43.061Z peitao-ThinkPad-X40 myproc 3431 L7 - : 4D 4E 4F 50 51 52 53 4B 55 56 57 58
 59 5A 21 40|MNOPQRSKUVWXYZ!@
<135>1 2017-04-25T23:32:43.061Z peitao-ThinkPad-X40 myproc 3431 L7 - : 23 24 25 5E 26 2A 28 29
           |#$%^&*()
```

5.2. Remote Capture utility tools

Embedded C logging comes up with remote UDP capture tools : eTrace for logging Linux and eTrace for logging Windows. These tools have a console window to real-time display colorful messages from target machine, meanwhile at the background save same messages to the local files . These tools are especially useful to capture undetectable errors, saying system corrupt over the long time.

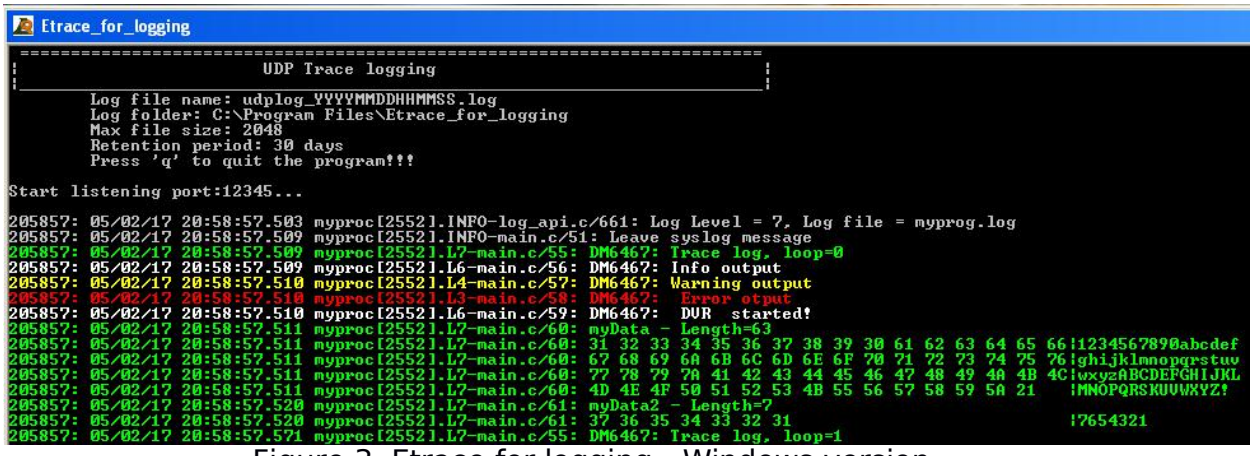

Figure 3. Etrace for logging - Windows version

Both tools (either Windows or Linux) have same command syntax:

Usage: **etrace port <debugFolder> <logFileSize> <retention days>**

- **Port** - UDP listen port (mandatory)
- **DebugFolder** - log file destination folder, default: current folder (option parameter)
- **LogFileSize** - Log file size, a new file will be created if over this size (option parameter)
- **Retention days** - the log files over this retention period will be deleted. (option parameter)

The nomenclature of log file name is “udplog_yyyymmddHHMMSS.log”.

To capture remote logging successfully, please check:

1. Embedded c logging setting on target machine and make sure UDP desitinaton is enabled , UDP host machine (that capture tool runs on) and UDP port are matched.
2. There is no firewall between target and UDP host machine, recommend to set UDP port over 1024
3. Target and UDP host machine are in the same local network, avoid to use VPN

6. 6 APPENDIX

A. Embedded C Logging sample file

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "log_api.h"
int main(int argc, char **argv)
{
        int debug_flag=0;
        int iResult = 0;
        char* myData=
"1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSKUVWXYZ!@#$%^&*()";
        /*Initialize Log Apl */
        iResult = LogApi_Init( "config_log.conf");
        if (iResult != TRUE)
        {
        fprintf(stderr,"Failed to initialize log library,exit!\n");
        exit (-1);
        }

        LogApp_Debug("DM6467: Trace log, loop=%d",debug_flag);
        LogApp_Info("DM6467: Info output");
        LogApp_Warn("DM6467: Warning output");
        LogApp_Error("DM6467: Error otput");
        LogApp_Info("DM6467: DVR started!");
        LogApp_HexDump_Debug("myData",myData, strlen(myData));
```

```
        printf_error("DM6467 --- printing error!");
        printf_warn("DM6467 --- printing warning!");
        printf_debug("DM6467 --- printing debug!");
        usleep(500000);
        /* Exit logApi and release resource*/
        LogApi_Term();
        return 0;
}
```

If Embedded C logging also ships with its library , the user can include log_api.h and link its library to the system. The following is a simple makefile to link embedded c logging library.

B Makefile Sample

```
# Filename: Makefile

CC=gcc

.PHONY: all

all: myproc

#liblog.a is the dependency for the executable

myproc: main.o liblog.a

$(CC) -lm -o myproc main.o -L. -llog

main.o: main.c log_api.h

$(CC) -O -c main.c

log_serial.o: log_serial.c log_serial.h

$(CC) -O -c log_serial.c

log_udp.o: log_udp.c log_udp.h

$(CC) -O -c log_udp.c

log_api.o: log_api.c log_api.h

$(CC) -O -c log_api.c

#let's link library files into a static library

liblog.a: log_api.o log_serial.o log_udp.o

ar rcs liblog.a log_api.o log_serial.o log_udp.o
```

```
libs: liblog.a

clean:

rm -f myproc *.o *.a *.gch #This way is cleaner than your clean
```

C Log configuration example

To the first time user who uses Embedded C logging, a log configuration file has to be created first, which can be put the same folder of running program: (relative running program, or other folder such as ../../config_log.conf)

config_log.conf

```
# Set log output destination. (bitwise)
# 0 : stdin (unused)
# 1 : stdout bit 0:
# 2 : stderr bit 1:
# 3 : (unused)
# 4 : file ( application log file) - bit 2:
# 5 or 6: console and log
# 7 : (unused)
# 8 : Serial port : bit 3
# 9 -10 : console and serial port
# 16 - UDP : bit 4:
# Available values: 1 - 30
#   22: output to stderr, file and remote host
destination = 22

# Enable/Disable syslog log. 1-enable, 0-disable
# To enable syslog, you should ensure   that your syslogd daemon is running for
#   syslog messages to be   correctly handled.
syslog_enable = 1

# sys log facilities for remote logging please refer RF3164
# This parameter is used to calculate syslog priority
# e.g. if facility=0 ( LOG_LOCAL0) and log_level =2
# priority of syslog = (16<<3) + LOGAPP_DEBUG(7) = 135
# log output over application UDP:
# <134>Apr 22 17:56:18 ThinkPad-X40 myproc[2541]: L6: DM6467: Info output
# 0: LOG_LOCAL0: (16<<3)
# 1; LOG_LOCAL1: (17<<3)
# 2: LOG_LOCAL2: (18<<3)
# 3: LOG_LOCAL3: (19<<3)
# 4: LOG_LOCAL4: (20<<3)
# 5: LOG_LOCAL5: (21<<3)
# 6: LOG_LOCAL6: (22<<3)
# 7: LOG_LOCAL7: (23<<3)
facility = 0; #LOG_LOCAL0

# Debug level from -5 (SYS_LOG_PNC) to 7 (LOGAPP_DEBUG)
# SYS_LOG_PNC = -5 -> LOG_ERR (syslog)
# SYS_LOG_FAT = -4 -> LOG_ERR (syslog)
# SYS_LOG_ERR = -3 -> LOG_ERR (syslog)
```

```
# SYS_LOG_WAR = -2 -> LOG_WARNING (syslog)
# SYS_LOG_INFO = -1 -> LOG_INFO (syslog)
#
# App log Local syslog Remote syslog
# LOGAPP_EMERG = 0 -> L0 -> LOG_EMERG =0
# LOGAPP_ALERT = 1 -> L1 -> LOG_ALERT =1
# LOGAPP_CRIT = 2 -> L2 -> LOG_CRIT =2
# LOGAPP_ERROR = 3 -> L3 -> LOG_ERR = 3
# LOGAPP_WARN = 4 -> L4 -> LOG_WARN = 4
# LOGAPP_NOTICE = 5 -> L5 -> LOG_NOTICE =5
# LOGAPP_INFO = 6 -> L6 -> LOG_INFO = 6
# LOGAPP_DEBUG = 7 -> L7 -> LOG_TRACE = 7
# Note:
# log_level < 0 ( disable application log)
# if syslog_enable =1,only has syslog.
# log_level > 0 ( enable application log),
# if syslog_enable =1, all local syslog will
# be logged as severity "Lx".e.g.
# Apr 22 18:08:14 peitao-ThinkPad-X40 myproc.log[2637]: [L6] DM6467: Info output
log_level = 7

# Debugging log file name (relative to running folder)
log_filename = myprog.log

#number of files to rotate
number_of_files = 9

#Max size of one logfile (in KB)
max_logfile_size = 50

# Configuration parameters for SERIAL destination (destination parameter bit 3)
# Serial communication port: such as /dev/ttyS0 or /dev/ttyUSB0
serial_com = /dev/tty2

# Configuration parameters for UDP destination (destination parameter bit4)
udp_host = 127.0.0.1
udp_port = 12345

# Configure log message format:
# 0: Time stamp mode: mmm DD %H:%M:%S, such as Aug 24 05:14:15
#       e.g.
#       04/13/10 15:55:46.971 myproc[1518].L7-main.c/53 [DM6467: debug log]
# 1: Runtime mode: elaspe time ( sec: msec ) from programing running
#       e.g.
#       03.375 myproc[1518].DB7-main.c/53: DM6467: debug log
#
# 2: syslog mode:RFC3164
#       <priority>Mmm DD HH:MM:SS host tag[pid]: msg
#       e.g:
#       <134>Apr 22 17:56:18 ThinkPad-X40 myproc[2541]: L6: DM6467: Info output
# 3: syslog mode: RFC5424
#       <priority>version YYYY-MM-DDTHH:MM:SS.MMMZ host proc pid msgid
structured_data msg
#       e.g.
```

```
#       <131>1 2017-04-22T18:02:31.824Z ThinkPad-X40 myproc 2580 L3 - : DM6467: Error
otput
# Note: When do remote logging over UDP interface, log_format=0 or 1 send messages with
# encoded color information so that it can display different color messages on
# remote PC tool It still be readable with third party UDP trace tool.
#
# To send messages to remote syslog server over UDP, if local syslog daemon is disabled,
# please choose log_format =2 or 3, RFC3164 or RFC5424;
log_format = 3
```