# @嵌入式软件 --- 日志接口规范 @

当前版本: V0.5

最后修改: 2011-05-23 08:57:41

## 说明:

本文档规范了嵌入式软件编写库和应用代码时应使用的一致性日志接口,

1~4 节说明了日志库提供的接口和具体的使用方法,

5~12 节说明了日志接口内容及实现原理等.

## 1.日志库接口定义

日志库(libxeLOG)提供的接口按照功能可以划分为如下内容:

### (1) 日志记录接口

级别 0 - xeLOG_EMERG(fmt, args...)

级别 1 - xeLOG_CRIT(fmt, args...)

级别 2 - xeLOG_ALERT(fmt, args...)

级别 3 - xeLOG_ERR(fmt, args...)

级别 4 - xeLOG_WARNING(fmt, args...)

级别 5 - xeLOG_NOTICE(fmt, args...)

级别 6 - xeLOG_INFO(fmt, args...)

级别 7 - xeLOG_DEBUG(fmt, args...)

注: 各级别的详细说明见第5节.

### (2) 日志记录扩展接口

```
#define DBG(fmt, args...)    xeLOG_DEBUG(fmt, ##args)
#define eprintf(fmt, args...) xeLOG_ERR(fmt, ##args)
#define dprintf(fmt, args...) xeLOG_DEBUG(fmt, ##args)
```

注: 日志内容的详细说明见第6节.

### （3）日志库初始化接口

本地记录方式:

```
int init_xelog_local(const init_xelog_local_t *ix_p);
int deinit_xelog_local(void);
```

远程记录方式:

```
int init_xelog_network(const init_xelog_remote_t *ix_p);
int deinit_xelog_network(void);
```

注：本地记录、远程记录方式详细说明见第9节.

### （4）日志查询接口

A.启动查询:

```
int xeLOG_query(const xelog_query_req_t *req);
```

B.读取日志,每次读取最多返回 128 条

```
int xeLOG_read( xelog_query_rps_t **rps);
```

C.关闭查询:

```
void xeLOG_close();
```

注：libxeLOG 提供日志查询功能，详细说明见第 10 节。

### （5）日志订阅接口

```
int xeLOG_subscribe(xelog_subscribe_req_t *req);
```

注：libxeLOG提供日志订阅功能，详细说明见第11节。

### （6）日志级别接口

```
int xeLOG_level_get(void);
int xeLOG_level_set(int new_level);
```

注：libxeLOG提供日志订阅功能，详细说明见第12节。

## 2.日志库相关文件存放位置

### （1）头文件

xelog.h 和 xeLOG.h,其中 xeLOG.h 是供各模块定义内部日志头文件的模板，不是必须的,若自己实现或用其他文件名，需在文件首部包含如下内容：

```
#ifdef CONFIG_USE_EXTERNAL_XELOG
#include <xelog.h>
#endif
```

（2）库文件

LOCAL 版本---->libxeLOG.***_LOCAL.a

REMOTE 版本--->libxeLOG.***_REMOTE.so

其中***代表目标平台、产品名称、版本号和发布日期.

各模块需使用的版本参见“使用示例”.

（3）发布时存放位置

svn 上以下两个路径都可以获取到头文件和库文件,两处发布的内容是相同的。

svn://svnserver/libbase/libxeLOG/release/

svn://svnserver/xeMODEL-core.NG/trunk/base/libxeLOG/

# 3.日志接口使用方法

应按照以下步骤使用日志接口:

（1）指定日志使用模式: 由宏 CONFIG_USE_EXTERNAL_XELOG 确定,可在 Makefile 中定义,编译时定义了这个宏, 则使用全局模式, 未定义则使用局部模式.

注：日志使用模式详细说明见第 8 节.

（2）指定日志记录方式: 局部模式下不需要该步设置, 全局模式下需要由宏 NO_LOG/XELOG_2_LOCAL/XELOG_2_REMOTE/XELOG_2_CONSOLE/XELOG_2_SYSLOG 确定,可以在 Makefile 中定义, 或直接在 xelog.h 中定义.

注：日志记录方式详细说明见第 9 节.

（3）指定模块名: 用于区分日志信息由谁产生,局部模式下可自定义,全局模式下由宏 XELOG_PREFIX 确定,可以在 Makefile 中定义, 或直接在 xelog.h 中定义.目前 libxeLOG 支持的模块名见下表.

| | | | |
|---|---|---|---|
| 基础库 | libxeCEO | libxeAA | libxeSOE |
| | LibxeSAP | LibxeSUCK | LibxePTZ |
| | LibxeLOG | LibxePM | LibxeSIS |
| | LibxeDavinci | libxeCONF | core_other |
| | app_S2506A | app_SIPUA | app_hxGUI |

| | | | |
|---|---|---|---|
| 应用 | app_hxWEB | app_other | |
| 其他 | kernel | syslog | |
| | libhxSDK | ramdisk | |

（4）初始化: 该步骤不是必须的，各模块是否需要初始化可见使用示例.

（5）订阅: 该步骤不是必须的，各模块可根据自身需要决定是否订阅.

（6）调用日志接口.

# 4.使用示例

## (1)使用示例 1---网络摄像机

编码板 (TARGET.mk_K2/X86_SIMULATE.HDTI64E0.K2-C6B0xx-x)

各模块使用方法如下:

1）核心模块:

a. 在 Makefile 中定义使用全局模式: CFLAGS +=
-DCONFIG_USE_EXTERNAL_XELOG=1

b. 链接静态库: libxeLOG.***_LOCAL.a

c. 包含头文件: #include “xelog.h”

d. 在 xelog,h 中定义记录方式: #define XELOG_2_LOCAL 1 (xelog.h)

e. 在 xelog,h 中指定模块名称: #define XELOG_PREFIX "core_other" (xelog.h)

f. 初始化: 使用接口 init_xelog_local(const init_xelog_local_t *ix_p)初始化,
其中初始化结构体如下: typedef struct init_xelog_local

```
{
  int  local_port;
  int  be_syslog;
  int  be_klog;
  int  log_level;
  char path[128];
  int  num;
```

```
        int  size;
    }init_xelog_local_t;
```

各成员含义如下:

local_port：是否接收远程日志,0-不接收,1-接收(此应用中必须为 1)

be_syslog：是否接管本地系统日志,0-不接管,1-接管

be_klog：是否接管本地内核日志,0-不接管,1-接管

log_level:日志记录级别

path:日志存储路径

num:日志文件数目　　size:每个日志文件的大小(以字节为单位)

g. 注册回调函数.

h. 使用日志记录 API.

i. 程序结束前反初始化 deinit_xelog_local().

2）基础库，以 libxeSAP 为例

a. 在 Makefile 中定义使用全局模式: CFLAGS +=

-DCONFIG_USE_EXTERNAL_XELOG=1

b. 链接静态库：libxeLOG.***_LOCAL.a

c. 包含头文件: #include “xelog.h”

e. 在 xelog,h 中定义记录方式: #define XELOG_2_LOCAL　　1

f. 在 xelog,h 中定义模块名称: #define XELOG_PREFIX　"libxeSAP"

g. 使用日志记录 API.

1）hxSDK

a. 在 Makefile 中定义使用全局模式: CFLAGS +=

-DCONFIG_USE_EXTERNAL_XELOG=1

b. 链接动态库: libxeLOG.***_REMOTE.so

c. 包含头文件: #include “xelog.h”

d. 在 xelog,h 中定义记录方式:　#define XELOG_2_REMOTE　　1

e. 在 xelog,h 中定义模块名称:　#define XELOG_PREFIX　　"libhxSDK"

f. 使用日志记录 API.

4) 应用模块，为 hxWEB 为例：

a. 在 Makefile 中定义使用全局模式: CFLAGS +=

-DCONFIG_USE_EXTERNAL_XELOG=1

b. 链接动态库: libxeLOG.***_REMOTE.so

c. 包含头文件: #include “xelog.h”

d. 在 xelog,h 中定义记录方式: #define XELOG_2_REMOTE 1

e. 在 xelog,h 中定义模块名称: #define XELOG_PREFIX "app_hxWEB"

f. 从 hxSDK 获取日志级别，并赋值给全局变量 log_level

g. 使用日志记录 API.

## (2)使用示例 2---网络视频服务器

### A. CPU 板 (TARGET.mk_K2/X86_SIMULATE.HDMA61CP.K2-S6A120)

各模块使用方法与使用示例 1 相同。

### B.编码板 (TARGET.mk_K2/X86_SIMULATE.HDTI64E1.K2-S6A120)

各模块使用方法如下：

1) 核心模块:

a. 在 Makefile 中定义使用全局模式: CFLAGS +=
-DCONFIG_USE_EXTERNAL_XELOG=1

b. 接静态库: libxeLOG.***_REMOTE.so

c. 包含头文件: #include “xelog.h”

d. 在 xelog,h 中定义记录方式: #define XELOG_2_REMOTE 1

e. 在 xelog,h 中定义模块名称: #define XELOG_PREFIX "core_other"

f. 初始化: 使用接口 init_xelog_network(const init_xelog_remote_t *ix_p)初始化,
初始化结构体如下: **typedef struct** init_xelog_remote

```
{
        int  log_level;
        int  be_syslog;
        int  be_klog;
        char remoteIP[32];
        int  remotePort;
}init_xelog_remote_t;
```

各成员含义如下:

log_level:日志记录级别

be_syslog：是否接管本地系统日志,0-不接管,1-接管

be_klog：是否接管本地内核日志,0-不接管,1-接管

remoteIP: 远程日志系统 IP 地址

remotePort: 远程日志系统接收端口，必须为 514

g. 使用日志记录 API.

h. 程序结束前反初始化 deinit_xelog_network().

2) 基础库，以 libxeSAP 为例：

a. 在 Makefile 中定义使用全局模式: CFLAGS += -DCONFIG_USE_EXTERNAL_XELOG=1

b. 链接静态库: libxeLOG.***_REMOTE.so

c. 包含头文件: #include “xelog.h”

d. 在 xelog,h 中定义记录方式: #define XELOG_2_REMOTE 1

e. 在 xelog,h 中定义模块名称: #define XELOG_PREFIX "libxeSAP"

f.使用日志记录 API.

3) hxSDK

a. 在 Makefile 中定义使用全局模式: CFLAGS += -DCONFIG_USE_EXTERNAL_XELOG=1

b. 链接静态库: libxeLOG.***_REMOTE.so

c. 包含头文件: #include “xelog.h”

d. 在 xelog,h 中定义记录方式: #define XELOG_2_REMOTE 1

e. 在 xelog,h 中定义模块名称: #define XELOG_PREFIX "libhxSDK"

f. 使用日志记录 API.

4) 应用模块，为 hxWEB 为例：

a. 在 Makefile 中定义使用全局模式: CFLAGS += -DCONFIG_USE_EXTERNAL_XELOG=1

b. 链接静态库: libxeLOG.***_REMOTE.so

c. 包含头文件: #include “xelog.h”

d. 在 xelog,h 中定义记录方式: #define XELOG_2_REMOTE 1

e. 在 xelog,h 中定义模块名称: #define XELOG_PREFIX "libxeSAP"

f. 从 hxSDK 获取日志级别，并赋值给全局变量 log_level

g. 使用日志记录 API.

### (3)使用示例 3---交通外设控制机

主控板(TARGET.mk_K2/X86_SIMULATE.HDARMSMT.K2-T6A00)

各模块使用方法与使用示例 1 相同。

## 5.日志级别说明

日志分 8 个级别，当需记录的日志的等级<=当前设定的日志记录等级时才记录日志。

级别 0: 用于紧急事件消息，它们一般是系统崩溃之前的提示消息.

级别 1: 用于立即采取动作的情况.

级别 2: 临界状态，通常涉及严重的硬件或软件操作失败.

级别 3: 用于报告错误状态, 特别是系统调用/C 库函数调用失败,
此时还需要记录错误代码及错误信息.

级别 4: 对可能出现问题的情况进行警告，但这类情况通常不会对系统造成严重问题.

级别 5: 有必要进行提示的正常情形, 许多与安全相关的状况用这个级别进行汇报.

级别 6: 提示信息, 例如: 操作完成, 状态发生了变化, 当一个客户端连接上来时.

级别 7: 调试信息, 此信息供开发人员跟踪程序运行轨迹.

## 6.日志内容说明

(1)日志接口自动添加的内容: 时间,级别,模块名,源代码文件名,行号以及换行符('\n').

(2)日志接口参数: 所有日志接口都需要相同的形参 fmt,args...,
fmt 确定日志信息的格式, args...确定具体内容,
用法同 printf(const char *format, ...)*(man 3 printf)*.
日志信息以行为单位进行记录, 会自动添加换行符,因此在 fmt 中无需再额外指定.

(3)日志接口 DBG(): 是一个特殊的接口, 可以用于调试阶段,
编译时若未定义 NO_DEBUG 宏, 等效于直接调用 xeLOG_DEBUG()日志接口,
编译时若定义了 NO_DEBUG 宏, 所有调用 DBG()接口的代码都不会被编译.

(4)日志接口 eprintf()：xeLOG_ERR()接口的别名.

(5)日志接口 dprintf()：xeLOG_DEBUG()接口的别名.

(6)在极其特殊的情况下, 编译时通过定义 NO_LOG 宏关闭所有的日志,
即所有调用日志接口的地方都不会被编译。

# 7.日志存储介质

目前支持普通文件和 Flash 记录, 发布 libxeLOG.a 时,介质已确定,

## (1)记录到文件

此方式下, 日志信息记录到文本文件中, 使用文本编辑器可以查看日志信息。使用者需要在调用 init_xelog_local()时指定目的文件路径,以及文件数目、每个文件的大小(以字节为单位). 若目的文件路径为空，则直接将日志信息打印到标准输出.

当每个日志文件都写满时,从旧到新开始覆盖。

## (2)记录到 Flash

此方式下, 日志信息记录到 flash 中,需要使用工具查询。 使用者需要在调用 init_xelog_local()时指定目的设备文件名.

# 8.日志使用模式

日志接口的使用者通过宏 CONFIG_USE_EXTERNAL_XELOG 来决定使用哪种模式, 编译时若定义了该宏, 则使用全局模式, 编译时若未定义则使用局部模式。

## (1)全局模式

这种模式下日志接口的实现依赖于 libxeLOG 日志基础库。

使用者需要在调用日志接口的源文件中包含 xelog.h,并指定静态链接库 libxeLOG.a.

## (2)局部模式

这种模式下日志接口由使用者自己实现,但必须保持日志接口一致.

使用者需要增加变量${module_name}_log_level 来设定日志级别, 例如 hxSDK 在默认情况下使用内部日志记录方式, 此时需要在内部头文件或 c 文件中增加如下定义:

```
#ifdef CONFIG_USE_EXTERNAL_XELOG
#include <xelog.h>
#else
extern int hxsdk_log_level;
#include <stdio.h>
#endif

#ifndef MSG_PREFIX
```

```
#define MSG_PREFIX "hxSDK"
#endif//MSG_PREFIX

#ifndef xeLOG_EMERG
#define xeLOG_EMERG(fmt, args...) \
if(hxsdk_log_level>=(0)) { \
  fprintf(stderr, MSG_PREFIX"(%s.%d.EMERG): "fmt"\n",__FILE__,__LINE__, ##args); \
  fflush(stderr); \
}
#endif//xeLOG_EMERG

#ifndef xeLOG_CRIT
#define xeLOG_CRIT(fmt, args...) \
  if(hxsdk_log_level>=(1)) { \
    fprintf(stderr, MSG_PREFIX"(%s.%d.CRIT): "fmt"\n",__FILE__,__LINE__, ##args); \
    fflush(stderr); \
  }
#endif//xeLOG_CRIT

#ifndef xeLOG_ALERT
#define xeLOG_ALERT(fmt, args...) \
  if(hxsdk_log_level>=(2)) { \
    fprintf(stderr, MSG_PREFIX"(%s.%d.ALERT): "fmt"\n",__FILE__,__LINE__, ##args); \
    fflush(stderr); \
  }
#endif//xeLOG_ALERT

#ifndef xeLOG_ERR
#define xeLOG_ERR(fmt, args...) \
  if(hxsdk_log_level>=(3)) { \
          fprintf(stderr, MSG_PREFIX"(%s.%d.ERR): "fmt"\n",__FILE__,__LINE__, ##args); \
          fflush(stderr); \
  }
#endif

#ifndef xeLOG_WARNING
#define xeLOG_WARNING(fmt, args...) \
```

```
  if(hxsdk_log_level>=(4)) { \
    fprintf(stderr, MSG_PREFIX"(%s.%d.WARNING): "fmt"\n",__FILE__,__LINE__, ##args); \
    fflush(stderr); \
  }
#endif//xeLOG_WARNING

#ifndef xeLOG_NOTICE
#define xeLOG_NOTICE(fmt, args...) \
  if(hxsdk_log_level>=(5)) { \
    fprintf(stderr, MSG_PREFIX"(%s.%d.NOTICE): "fmt"\n",__FILE__,__LINE__, ##args); \
    fflush(stderr); \
  }
#endif//xeLOG_NOTICE

#ifndef xeLOG_INFO
#define xeLOG_INFO(fmt, args...) \
  if(hxsdk_log_level>=(6)) { \
    fprintf(stdout, MSG_PREFIX"(%s.%d.INFO): "fmt"\n",__FILE__,__LINE__, ##args); \
    fflush(stdout); \
  }
#endif//xeLOG_INFO

#ifndef xeLOG_DEBUG
#define xeLOG_DEBUG(fmt, args...) \
  if(hxsdk_log_level>=(7)) { \
    fprintf(stdout, MSG_PREFIX"(%s.%d.DBG): "fmt"\n",__FILE__,__LINE__, ##args); \
    fflush(stdout); \
  }
#endif//

#ifndef DBG
#define DBG(fmt, args...) xeLOG_DEBUG(fmt, ##args)
#endif//DBG

#ifndef eprintf
#define eprintf(fmt, args...) xeLOG_ERR(fmt, ##args)
#endif//eprintf
```

```
#ifndef dprintf
#define dprintf(fmt, args...) xeLOG_DEBUG(fmt, ##args)
#endif//dprintf
```

## 9.日志记录方式

目前实现的日志记录方式是互斥的, 只能使用其中一种方式.

### (1)本地记录

在编译时定义 XELOG_2_LOCAL 宏, 或直接在 xelog.h 中定义.

此方式下, 日志信息记录在本地, 调用日志接口之前需要调用 init_xelog_local()进行初始化,程序退出前需要调用 deinit_xelog_local()反初始化.

初始化的内容有:

path： 本地日志文件路径或 flash 设备文件名

num ： 日志文件数目,即由几个文件存储日志

size： 每个日志文件的大小,以字节为单位

local_port： 本地接收端口, 接收远程日志时必须为 514, 不接收时为 0

log_level ： 日志记录级别

### (2)远程记录

在编译时定义 XELOG_2_REMOTE 宏, 或直接在 xelog.h 中定义.

此方式下, 日志信息将以 UDP 方式发送到远程日志系统, 调用日志接口之前需要调用 init_xelog_network()进行初始化, 程序退出前需调用 deinit_xelog_network()反初始化.

初始化的内容有:

remoteIP ：日志接收者 IP 地址

remotePort：日志接收端口, 必须为 514

log_level ：日志记录级别

### (3)标准输出/标准错误

在编译时定义 XELOG_2_CONSOLE 宏, 或直接在 xelog.h 中定义.

可在调用日志接口前,修改 stdout/stderr,使日志信息写往自定义的文件或网络目的地.

### (4)调用 syslog

在编译时定义 XELOG_2_SYSLOG 宏, 或直接在 xelog.h 中定义.

调用 syslog()的日志信息会发送到主机的 syslogd 服务程序, 此程序可配置成根据不同日志级别写往不同日志文件, 及发送到其它主机的 syslogd 服务程序. 具体可通过 man syslogd 和 man syslog.conf 获得其手册.

### (5)不记录日志

在编译时定义 NO_LOG 宏, 或直接在 xelog.h 中定义, 则不记录日志.

## 10.日志查询

libxeLOG 提供日志查询接口,可根据时间、模块、级别查询日志,

支持多模块、多级别查询, 结果以结构体返回, 接口及相关结构如下:

(1)接口

**A. int xeLOG_query**(**const** xelog_query_req_t *req);

启动查询,初始化查询相关参数,返回值说明:

0: 启动成功, 可以进行读取操作.

1: 有其他查询操作在进行,需要等其结束后再次查询(加入了单线程限制).

-1: 启动失败.

**B. int xeLOG_read**(xelog_query_rps_t **rps);

读取日志信息, 需要在启动查询成功后进行, 查询结果的内存空间由库分配，并由库释放(xeLOG_close)，用户不需要再次释放。

返回值说明:

>=0: 读取到的日志信息条数,每次读取最多 128 条.

-1: 读取失败.

说明:应在循环中多次读取,直到返回值为 0 时,查询结束.

**C. void xeLOG_close**();

关闭查询,释放查询结果链表,不需要再读取时必须进行该操作, 否则其他查询任务将被阻塞.

(2)时间

```
typedef struct xelog_datetime
{
```

```
        int year;
        int month;
        int day;
        int hour;
        int minute;
        int second;
    } xelog_datetime_t;
```

(3)查询条件结构体

```
    typedef struct xelog_query_request
    {
        xelog_datetime_t stime;//开始时间

        xelog_datetime_t etime;//结束时间

        unsigned long level;//级别, 支持单个/多个级别, 宏定义见“附件1”.

        unsigned long module;//模块,支持单个/多个级别, 宏定义见“附件2”.
    } xelog_query_req_t;
```

(4)查询结果结构体

```
    typedef struct xelog_query_response
    {
        int level;//日志级别

        char module[16];//模块名称

        char data[256];//日志内容

        xelog_datetime_t tick;//日志发生时间

        struct xelog_query_response *next;//下一条
    } xelog_query_rps_t;
```

## 11.日志订阅

日志记录多作为出现问题后分析问题来源的依据,但对于一些重要事件,应用程序希望在问题发生时立即进行处理, 日志订阅即用于实现该功能,支持多模块、多级别订阅.

应用程序根据自己的处理逻辑定义事件处理函数, 并使用日志订阅接口进行订阅, 指定当某模块的某个级别的日志发生时执行哪个处理函数,日志库会在该事件发生时执行处理函数,接口及相关结构如下:

(1)接口

```
int xeLOG_subscribe(xelog_subscribe_req_t *req);
```

(2)订阅条件结构体

```
typedef struct xelog_subscribe_request
{
  unsigned long module;//订阅那个或哪些模块
  unsigned long level; //订阅那个或哪些模块
  xelog_subscribe_f sub_f;//事件处理回调函数
} xelog_subscribe_req_t;
```

模块和级别由附件 1、2 中的宏确定，当对某些模块/级别采取相同的处理方式时，可用 XELOG_MODULE_KERNEL | XELOG_MODULE_CORE 的形式定义 module，级别类似，即采用位或的方式定义多模块、多级别。

(3)事件处理回调函数

```
typedef int (*xelog_subscribe_f)(xelog_subscribe_rps_t *rps);
```

其中事件发生时返回信息的结构体为:

```
typedef struct xelog_subscribe_response
{
  int level; //级别
  char module[16];//模块名称
  char message[256];//日志内容
} xelog_subscribe_rps_t;
```

## 12.日志级别

libxeLOG 提供日志级别获取、设置接口,定义如下:

(1)获取当前日志级别

```
int xeLOG_level_get(void);
```

返回值说明:

>=0: 当前的日志级别.

-1: 获取失败.

(2)设置日志级别

```
int xeLOG_level_set(int new_level);
```

参数说明:

new_level:要设置的日志级别.

返回值说明:

0: 设置成功.

-1: 设置失败.

附件 **1**－日志级别宏定义

```
/*log level xxx */
#define XELOG_LEVEL_EMERG               0x00000001UL
#define XELOG_LEVEL_CRIT            0x00000002UL
#define XELOG_LEVEL_ALERT           0x00000004UL
#define XELOG_LEVEL_ERR                 0x00000008UL
#define XELOG_LEVEL_WARNING             0x00000010UL
#define XELOG_LEVEL_NOTICE          0x00000020UL
#define XELOG_LEVEL_INFO            0x00000040UL
#define XELOG_LEVEL_DEBUG           0x00000080UL

/*log level from 0 to N */
#define XELOG_LEVEL_0           XELOG_LEVEL_EMERG
#define XELOG_LEVEL_1           (XELOG_LEVEL_0 | XELOG_LEVEL_CRIT)
#define XELOG_LEVEL_2           (XELOG_LEVEL_1 | XELOG_LEVEL_ALERT)
#define XELOG_LEVEL_3           (XELOG_LEVEL_2 | XELOG_LEVEL_ERR)
#define XELOG_LEVEL_4           (XELOG_LEVEL_3 | XELOG_LEVEL_WARNING)
#define XELOG_LEVEL_5           (XELOG_LEVEL_4 | XELOG_LEVEL_NOTICE)
#define XELOG_LEVEL_6           (XELOG_LEVEL_5 | XELOG_LEVEL_INFO)
#define XELOG_LEVEL_7           (XELOG_LEVEL_6 | XELOG_LEVEL_DEBUG)
```

附件 **2**－日志模块宏定义

```
/*log module xxx */
#define XELOG_MODULE_NONE                       0x00000000UL

#define XELOG_MODULE_KERNEL_DM642           0x00000001UL
#define XELOG_MODULE_KERNEL_OS              0x00000002UL
#define XELOG_MODULE_KERNEL \
```

```
                (XELOG_MODULE_KERNEL_DM642
                                | XELOG_MODULE_KERNEL_OS)

#define XELOG_MODULE_LIB_CEO                0x00000010UL
#define XELOG_MODULE_LIB_AA                 0x00000020UL
#define XELOG_MODULE_LIB_SOE                0x00000040UL
#define XELOG_MODULE_LIB_SAP                0x00000080UL
#define XELOG_MODULE_LIB_SUCK               0x00000100UL
#define XELOG_MODULE_LIB_PTZ                0x00000200UL
#define XELOG_MODULE_LIB_LOG                0x00000400UL
#define XELOG_MODULE_LIB_PM                 0x00000800UL
#define XELOG_MODULE_LIB_SIS                0x00001000UL
#define XELOG_MODULE_LIB_DAVINCI            0x00002000UL
#define XELOG_MODULE_LIB_CONF               0x00004000UL
#define XELOG_MODULE_CORE_OTHER             0x00008000UL
#define XELOG_MODULE_CORE \
                ( XELOG_MODULE_LIB_CEO \
                        | XELOG_MODULE_LIB_AA \
                        | XELOG_MODULE_LIB_SOE \
                        | XELOG_MODULE_LIB_SAP \
                        | XELOG_MODULE_LIB_SUCK \
                        | XELOG_MODULE_LIB_PTZ \
                        | XELOG_MODULE_LIB_LOG \
                        | XELOG_MODULE_LIB_PM \
                        | XELOG_MODULE_LIB_SIS \
                        | XELOG_MODULE_LIB_DAVINCI \
                        | XELOG_MODULE_LIB_CONF\
                        | XELOG_MODULE_CORE_OTHER)

#define XELOG_MODULE_APP_S2506A             0x00010000UL
#define XELOG_MODULE_APP_SIPUA          0x00020000UL
#define XELOG_MODULE_APP_HXGUI          0x00040000UL
#define XELOG_MODULE_APP_HXWEB              0x00080000UL
#define XELOG_MODULE_APP_OTHER              0x00100000UL
```

```
#define XELOG_MODULE_APP  \
            (XELOG_MODULE_APP_S2506A \
                        | XELOG_MODULE_APP_SIPUA \
                        | XELOG_MODULE_APP_HXGUI \
                        | XELOG_MODULE_APP_HXWEB \
                        | XELOG_MODULE_APP_OTHER )

#define XELOG_MODULE_RAMDISK            0x01000000UL
#define XELOG_MODULE_SYSLOG             0x02000000UL
#define XELOG_MODULE_HXSDK              0x04000000UL

#define XELOG_MODULE_USER \
                ( XELOG_MODULE_RAMDISK \
                        | XELOG_MODULE_SYSLOG \
                        | XELOG_MODULE_HXSDK \
                        | XELOG_MODULE_CORE \
                        | XELOG_MODULE_APP)

#define XELOG_MODULE_ALL\
                (XELOG_MODULE_USER   | XELOG_MODULE_KERNEL)
```

# 更新说明:

V0.1 2008-07-15 wuhm <wuhm@hisome.com>

初始版本讨论稿.

V0.2 2010-11-09 renhw <renhw@hisome.com>

实现本地、远程记录方式,增加了使用示例、日志订阅等.

V0.3 2011-03-22 renhw <renhw@hisome.com>

修改日志查询接口, 加入日志读取接口、加入查询单线程限制.

V0.4 2011-04-21 renhw <renhw@hisome.com>

(1)增加接口 xeLOG_level_set/get(),用于动态设置、获取日志级别.

(2)修改初始化接口,加入获取当前日志文件存储位置的方法.

(3)修改反初始化接口,加入对缓存的日志信息的处理.

V0.5 2011-05-20 renhw <renhw@hisome.com>

(1)xelog.h 中增加 BUG 和 BUGMSG 定义。

(2)修改xeLOG_read()和xeLOG_close()接口，由库负责释放查询结果链表.