ICS 国际标准分类号
CCS 中国标准文献分类号

团　　体　　标　　准

T/CES XXX-XXXX

电力智能终端硬件抽象层接口技术规范

Technical specification for hardware abstraction layer software and interface of smart electric power terminal unit

XXXX-XX-XX 发布　　XXXX-XX-XX 实施

中国电工技术学会　发布

目　　次

前　言

本文件按照 GB/T 1.1—2020《标准化工作导则　第 1 部分：标准化文件的结构和起草规则》给出的规则起草。

请注意本文件的某些内容可能涉及专利，本文件的发布机构不承担识别这些专利的责任。

本文件由中国电工技术学会提出。

本文件由中国电工技术学会标准工作委员会能源智慧化工作组归口。

本文件起草单位：国网信息通信产业集团有限公司、四川中电启明星信息技术有限公司、中国电力科学研究院、国网智能电网研究院有限公司。

本文件主要起草人：吕东东、李温静、何明阳、李庆尧、佘文魁、张帅、李炳森、张冀川、孙浩洋、卜宪德。

本文件为首次发布。

电力智能终端硬件抽象层接口技术规范

1 范围

本文件规定了电力智能终端硬件抽象层接口调用框架、设计要求及接口定义。

本文件适用于电力智能终端（以下简称“终端”）硬件抽象层的开发和使用。

2 规范性引用文件

下列文件对于本文件的应用是必不可少的。凡是注日期的引用文件，仅注日期的版本适用于本文件。凡是不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 11457—2016 信息技术 软件工程术语

GB/T 16262.1—2006 信息技术 抽象语法记法（ASN.1） 第 1 部分：基本记法规范

GB/T 17626.30 电磁兼容 试验和测量技术 电能质量测试方法

GB/T 17966—2000 微处理器系统的二进制浮点运算

GB/T 20272—2019 信息安全技术 操作系统安全技术要求

DL/T 645—2007 多功能电能表通信协议

DL/T 790.6—2010 采用配电线载波系统的配电自动化 第 6 部分：A-XDR 编码规则

Q/GDW 10376.2—2019 电力用户用电信息采集系统通信协议 第 2 部分：集中器本地通信模块接口协议

Q/GDW 11778—2017 面向对象的用电信息数据交换协议

3 术语和定义

下列术语和定义适用于本文件。

3.1

硬件抽象层 hardware abstract layer

位于操作系统内核与硬件电路之间的接口层，其目的在于将硬件抽象化。它隐藏了特定平台的硬件接口细节,为操作系统提供虚拟硬件平台,使其具有硬件无关性,可在多种平台上进行移植。

3.2

文件系统 file system

操作系统用于明确磁盘或分区上的文件的方法和数据结构，既指磁盘上组织文件的方法，也指用于存储文件的磁盘或分区。

4 符号、代号和缩略语

下列符号、代号和缩略语适用于本文件。

API: 应用程序编程接口（application programming interface）

HAL：硬件抽象层（hardware abstract layer）

LED：发光二极管（light emitting diode）

RTC：实时时钟（real time clock）

USB：通用串行总线（universal serial bus）

WDT：看门狗（watch dog timer）

5 HAL 接口调用框架

HAL 接口调用框架要求如下：

a） 应作为设备驱动和系统调用的一个封装；
b） 应由系统接口、设备操作接口组成；
c） 应在操作系统中以动态库的形式为上层应用软件提供统一的调用接口，上层应用软件不必知道下层的具体实现,见图 1。

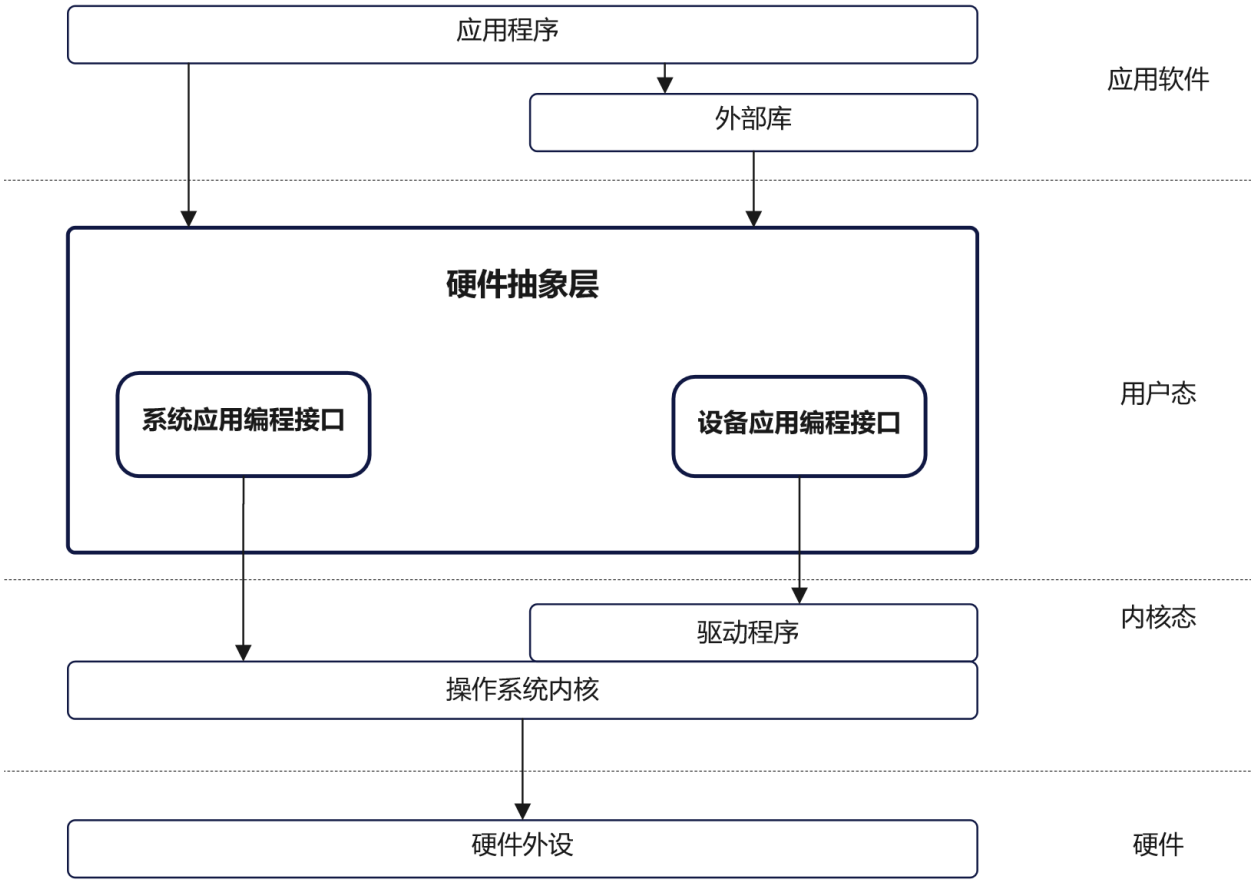

图1 HAL 接口调用框架图

6 HAL 设计要求

6.1 HAL 职责

HAL 应隐藏不同嵌入式操作系统和硬件设备的具体实现细节，为各类应用软件提供标准化调用接口。

6.2 HAL 设计原则

HAL 具体设计原则如下：

a） 应向前兼容，可扩增 HAL 的 API，但不能减少；
b） 应统一定义系统 API，适配各种嵌入式操作系统基础调用接口，满足上层应用软件开发需求；
c） 应统一定义设备操作 API，对应的驱动程序负责将这些操作调用映射到实际的硬件；
d） 应保持 API 风格一致；
e） 设备节点是对设备的抽象，应保证所有设备节点都以文件的形式存放在/dev 目录下，应用软件通过设备节点名称访问设备。

6.3 HAL 约束条件

HAL 具体约束条件如下：

a） 应以动态库的形式随系统发布；
b） 应保证设备驱动至少实现 open, close, read, write, ioctl 操作；
c） 设备节点所在路径应统一定义为：/dev/。

7 HAL 接口定义

7.1 HAL 对外接口定义

HAL 框架对外提供接口如下：

a） 应具备系统函数调用接口；
b） 应具备设备节点操作接口。

7.2 HAL 系统函数调用接口

7.2.1 内存管理

系统内存管理的接口见表 1，定义详情见附录 A.1。

表 1 系统内存管理接口列表

序 号	接 口 名 称	接 口 描 述
1	hal_malloc	内存分配
2	hal_realloc	内存调整
3	hal_free	内存释放

7.2.2 进程管理

系统进程管理的接口见表 2，定义详情见附录 A.2。

表 2 系统进程管理接口列表

序 号	接 口 名 称	接 口 描 述
1	hal_fork_create	创建进程
2	hal_fork_exit	终止进程
3	hal_fork_exec	运行可执行文件
4	hal_fork_getpid	获取进程标识
5	hal_fork_waitpid	等待进程终止

7.2.3 信号量

系统信号量的接口见表 3，定义详情见附录 A.3。

表 3 系统信号量接口列表

序 号	接 口 名 称	接 口 描 述
1	hal_semaphore_create	创建信号量
2	hal_semaphore_destroy	销毁信号量
3	hal_semaphore_post	发送信号量
4	hal_semaphore_wait	等待信号量

7.2.4 线程管理

系统线程管理的接口见表 4，定义详情见附录 A.4。

表 4　系统线程管理接口列表

序 号	接 口 名 称	接 口 描 述
1	hal_thread_create	创建线程
2	hal_thread_detach	设置线程分离
3	hal_thread_delete	删除线程

7.2.5　互斥锁

系统互斥锁的接口见表 5，定义详情见附录 A.5。

表 5　系统互斥锁接口列表

序 号	接 口 名 称	接 口 描 述
1	hal_mutex_create	创建互斥锁
2	hal_mutex_destroy	销毁互斥锁
3	hal_mutex_lock	锁上互斥锁
4	hal_mutex_unlock	解锁互斥锁

7.2.6　文件管理

系统文件管理的接口见表 6，定义详情见附录 A.6。

表 6　系统文件管理接口列表

序 号	接 口 名 称	接 口 描 述
1	hal_file_open	打开文件
2	hal_file_close	关闭文件
3	hal_file_read	读文件
4	hal_file_write	写文件
5	hal_file_seek	移动文件指针

7.2.7　时间管理

系统时间管理的接口见表 7，定义详情见附录 A.7。

表 7　系统时间管理接口列表

序 号	接 口 名 称	接 口 描 述
1	hal_timer_create	创建计时器
2	hal_timer_delete	删除计时器
3	hal_timer_start	启动计时器
4	hal_timer_stop	停止计时器
5	hal_get_time_ms	获取系统当前时间，单位毫秒
6	hal_sleep_ms	休眠，单位毫秒

7.2.8 套接字（socket）

套接字（socket）操作接口见表 8，定义详情见附录 A.8。

表 8 套接字（socket）操作接口列表

序 号	接 口 名 称	接 口 描 述
1	hal_socket_open	建立套接字
2	hal_socket_close	关闭套接字
3	hal_socket_write	向 socket 写数据
4	hal_socket_read	从 socket 读数据
5	hal_socket_connect	连接远程主机

7.2.9 系统应用

系统应用的接口见表 9，定义详情见附录 A.9。

表 9 系统应用接口列表

序 号	接 口 名 称	接 口 描 述
1	hal_snprintf	格式化字符串
2	hdl_printf	打印信息

7.3 HAL 设备节点操作接口

7.3.1 设备节点操作接口

设备节点操作接口见表 10。

表 10 设备节点操作接口列表

序 号	接 口 名 称	接 口 描 述
1	hal_device_open	打开设备节点
2	hal_device_close	关闭设备节点
3	hal_device_read	设备节点读数据
4	hal_device_write	设备节点写数据
5	hal_device_set	设置设备节点参数

7.3.2 设备节点命名和接口详情

7.3.2.1 RS485

设备节点名称为：

485-n:“/dev/ttyRSn”，*n* 为整数（0 - 255）。

接口定义详情见附录 A.10。

7.3.2.2 载波

设备节点名称为：

“/dev/ttyPLCn”，n 为整数（0 - 255）。

接口定义详情见附录 A.11。

7.3.2.3 4G

设备节点名称为：

“/dev/ttyLTEn”，n 为整数（0 - 255）。

接口定义详情见附录 A.12。

7.3.2.4 蓝牙

设备节点名称为：

“/dev/ttyBTn”，n 为整数（0 - 255）。

接口定义详情见附录 A.13。

7.3.2.5 LED

设备节点名称为：

“/dev/ledn”，n 为整数（0 - 255）。

接口定义详情见附录 A.14。

7.3.2.6 WDT

设备节点名称为：

“/dev/watchdog”。

接口定义详情见附录 A.15。

附 录 A
（资料性附录）
HAL 对外接口定义

A.1 内存管理接口定义

内存管理接口定义详情见表 A.1。

表 A.1 内存管理接口定义详情表

序号	接口名称	接口定义
1	内存分配	/** * @brief 内存分配 * @param[in] size: 分配大小 * @return 成功返回内存首地址；失败返回 NULL */ void *hal_malloc(unsigned int size);
2	内存调整	/** * @brief 内存调整 * @param[in] ptr: 内存地址 * @param[in] size: 调整大小 * @return 成功返回内存首地址；失败返回 NULL */ void *hal_realloc(void *ptr, unsigned int size);
3	内存调整	/** * @brief 内存释放 * @param[in] ptr: 内存地址 * @return 无 */ void hal_free(void *ptr);

A.2 进程管理接口定义

进程管理接口定义详情见表 A.2。

表 A.2 进程管理接口定义详情表

序号	接口名称	接口定义
1	创建进程	/** * @brief 创建进程 * @param[in] : 无 * @return 成功返回 0；失败返回错误码 */ pid_t hal_fork_create(void);
2	终止进程	/** * @brief 终止进程 * @param[in] status: 进程状态 * @return: 无 */ void hal_fork_exit(int status);

3	运行可执行文件	/** * @brief 运行可执行文件 * @param[in] path: 可执行文件路径 * @param[in] arg: 参数 * @param[in] ...: 参数 * @return 成功返回 0；失败返回错误码 */ int hal_fork_exec(const char * path,const char * arg, ...);
4	获取进程标识	/** * @brief 获取进程标识 * @param[in] : 无 * @return 成功返回当前进程标识；失败返回错误码 */ pid_t hal_fork_getpid(void);
5	等待进程终止	/** * @brief 等待进程终止 * @param[in] pid: 进程标识 * @param[in] status: 进程状态 * @param[in] options: 操作码 * @return 成功返回子进程标识；失败返回错误码 */ pid_t hal_fork_waitpid(pid_t pid,int * status,int options);

A.3 信号量接口定义

信号量接口定义详情见表 A.3。

表 A.3 信号量接口定义详情表

序号	接口名称	接口定义
1	创建信号量	/** * @brief 创建信号量 * @param[in] 无 * @return 成功返回信号量对象地址；失败返回 NULL */ void *hal_semaphore_create(void);
2	销毁信号量	/** * @brief 销毁信号量 * @param[in] sem:信号量对象 * @return :成功返回 0；失败返回错误码 */ int hal_semaphore_destroy(void *sem);
3	发送信号量	/** * @brief 发送信号量 * @param[in] sem:信号量对象 * @return :成功返回 0；失败返回错误码 */ int hal_semaphore_post(void *sem);

4	等待信号量	/** * @brief 等待信号量 * @param[in] sem:信号量对象 * @param[in] timeout_ms:超时时间 * @return:成功返回 0；失败返回错误码 */ int hal_semaphore_wait(void *sem, unsigned int timeout_ms);

A.4 线程管理接口定义

线程管理接口定义详情见表 A.4。

表 A.4 线程管理接口定义详情表

序号	接口名称	接口定义
1	创建线程	/** * @brief 创建线程 * @param[in] thread_handle:线程标识 * @param[in] work_routine:线程调用函数 * @param[in] arg:调用函数参数 * @return:成功返回 0；失败返回错误码 */ int hal_thread_create(void **thread_handle, void *(*work_routine)(void *), void *arg);
2	设置线程分离	/** * @brief 设置线程分离 * @param[in] thread_handle:线程标识 * @return:成功返回 0；失败返回错误码 */ int hal_thread_detach(void *thread_handle);
3	删除线程	/** * @brief 删除线程 * @param[in] thread_handle:线程标识 * @return:成功返回 0；失败返回错误码 */ int hal_thread_delete(void *thread_handle);

A.5 互斥锁接口定义

互斥锁接口定义详情见表 A.5。

表 A.5 互斥锁接口定义详情表

序号	接口名称	接口定义
1	创建互斥锁	/** * @brief 创建互斥锁 * @param[in] :无 * @return:成功返回互斥锁句柄；失败返回 NULL */ void *hal_mutex_create(void);

2	销毁互斥锁	/** * @brief 销毁互斥锁 * @param[in] mutex:互斥锁句柄 * @return:成功返回 0；失败返回错误码 */ int hal_mutex_destroy(void *mutex);
3	锁上互斥锁	/** * @brief 锁上互斥锁 * @param[in] mutex:互斥锁句柄 * @return:成功返回 0；失败返回错误码 */ int hal_mutex_lock(void *mutex);
4	解锁互斥锁	/** * @brief 解锁互斥锁 * @param[in] mutex:互斥锁句柄 * @return:成功返回 0；失败返回错误码 */ int hal_mutex_unlock(void *mutex);

A.6 文件管理接口定义

文件管理接口定义详情见表 A.6。

表 A.6 文件管理接口定义详情表

序号	接口名称	接口定义
1	打开文件	/** *@brief 字符串 mode 定义为文件打开形态 r 打开只读文件； r+ 打开可读写的文件； w 打开只写文件； w+ 打开可读写文件； a 以附加的方式打开只写文件； a+ 以附加方式打开可读写的文件； 上述的形态字符串都可以再加一个 b 字符，如 rb、w+b 或 ab＋等组合，加入 b 字符用来告诉函数库打开的文件为二进制文件，而非纯文字文件。 * @brief 打开文件 * @param[in] path:文件路径 * @param[in] mode:打开形态 * @return:成功返回文件指针；失败返回 NULL */ FILE *hal_file_open(const char *path,const char * mode);
2	关闭文件	/** * @brief 关闭文件 * @param[in] stream:文件指针 * @return:成功返回 0；失败返回错误码 */ int hal_file_close(FILE *stream);

3	读文件	/** * @brief 读文件 * @param[in] buff:读缓存 * @param[in] len:读最大字节数 * @param[in] stream:文件指针 * @return:成功返回读字节数；失败返回错误码 */ int hal_file_read(void *buff, int len, FILE *stream);
4	写文件	/** * @brief 写文件 * @param[in] buff:写缓存 * @param[in] len:写最大字节数 * @param[in] stream:文件指针 * @return:成功返回写字节数；失败返回错误码 */ int hal_file_write(void *buff, int len, FILE *stream);
5	移动文件指针	/** *@brief 参数 whence 定义为文件指针移动位置 SEEK_SET 从距文件开头 offset 位移量为新的读写位置； SEEK_CUR 以目前的读写位置往后增加 offset 个位移量； SEEK_END 将读写位置指向文件尾后再增加 offset 个位移量。 * @brief 移动文件指针 * @param[in] stream:文件指针 * @param[in] offset:移动数量 * @param[in] whence:移动位置 * @return:成功返回 0；失败返回错误码 */ int hal_file_seek(FILE * stream, int offset, int whence);

A.7 时间管理接口定义

时间管理接口定义详情见表 A.7。

表 A.7 时间管理接口定义详情表

序号	接口名称	接口定义
1	创建计时器	/** * @brief 创建计时器 * @param[in] func:回调函数 * @param[in] user_data:回调函数参数 * @return:成功返回定时器对象地址；失败返回 NULL */ void *hal_timer_create(void (*func)(void *), void *user_data);
2	删除计时器	/** * @brief 删除计时器 * @param[in] timer:定时器对象 * @return:成功返回 0；失败返回错误码 */ int hal_timer_delete(void *timer);

3	启动计时器	/** * @brief 启动计时器 * @param[in] timer:定时器对象 * @return:成功返回 0；失败返回错误码 */ int hal_timer_start(void *timer);
4	停止计时器	/** * @brief 停止计时器 * @param[in] timer:定时器对象 * @return:成功返回 0；失败返回错误码 */ int hal_timer_stop(void *timer);
5	获取系统当前时间	/** * @brief 获取系统当前时间，单位毫秒 * @param[in] :无 * @return:成功返回当前时间戳；失败返回错误码 */ int64 hal_get_time_ms(void);
6	休眠	/** * @brief 休眠，单位毫秒 * @param[in] ms:休眠时间 * @return:无 */ void hal_sleep_ms(unsigned int ms);

A.8 socket 套接字接口定义

socket 套接字接口定义详情见表 A.8。

表 A.8 socket 套接字接口定义详情表

序号	接口名称	接口定义
1	建立套接字	/** * @brief 建立套接字 * @param[in] :无 * @return:成功返回 socket 句柄；失败返回错误码 */ int hal_socket_open(void);
2	关闭套接字	/** * @brief 关闭套接字 * @param[in] sockfd:socket 句柄 * @return:成功返回 0；失败返回错误码 */ int hal_socket_close(int sockfd);
3	向 socket 写数据	/** * @brief 向 socket 写数据 * @param[in] sockfd:socket 句柄 * @param[in] send_buff:写数据缓存 * @param[in] send_len:写数据长度 * @return:成功返回写数据长度；失败返回错误码

4	从 socket 读数据	/** * @brief 从 socket 读数据 * @param[in] sockfd:socket 句柄 * @param[in] recv_buff:读数据缓存 * @param[in] recv_len:读数据长度 * @return:成功返回读数据长度；失败返回错误码 */ int hal_socket_read(int sockfd, unsigned char *recv_buff, unsigned int recv_len);
5	连接远程主机	/** * @brief 连接远程主机 * @param[in] sockfd:socket 句柄 * @param[in] server:主机地址 * @param[in] port:主机端口 * @return:成功返回 0；失败返回错误码 */ int hal_socket_connect(int sockfd, const char *server, const int port);

A.9 系统应用接口定义

系统应用接口定义详情见表 A.9。

表 A.9 系统应用接口定义详情表

序号	接口名称	接口定义
1	格式化字符串	/** * @brief 格式化字符串 * @param[in] str:目的字符串地址 * @param[in] len:写入的字符串长度 * @param[in] fmt:源字符串地址 * @param[in] ...:格式化内容 * @return:成功返回格式化字符串长度；失败返回错误码 */ int hal_snprintf(char *str, const int len, const char *fmt, ...);
2	打印信息	/** * @brief 打印信息 * @param[in] fmt:字符串格式 * @param[in] ...:打印内容 * @return:无 */ void hal_printf(const char *fmt, ...);

A.10 RS485 模块接口定义

RS485 模块接口定义详情见表 A.10。

表 A.10 RS485 模块接口定义详情表

序号	接口名称	接口定义

1	打开设备节点	/** *@brief 用于配置 int flags 的宏定义,不能同时出现但是必须有一种出现 O_RDWR： 只读模式 O_WRONLY: 只写模式 O_RDWR: 可读可写模式 * @brief 打开设备节点 * @param[in] dev_name: 设备节点名称 * @param[in] flags:打开配置标志，有只读；只写；读写模式 * @return 成功返回设备描述符；失败返回错误码 */ int hal_device_open(char * dev_name,int flags);
2	关闭设备节点	/** * @brief 关闭设备节点 * @param[in] fd:设备描述符 * @return:成功返回 0；失败返回错误码 */ int hal_device_close(int fd);
3	设备节点读数据	/** * @brief 读设备数据 * @param[in] fd: 设备描述符 * @param[out] buf :读缓存区 * @param[in] len: 缓存区长度 * @return 成功返回已读取字节长度；失败返回错误码 */ int hal_device_read(int fd, unsigned char *buf, int len);
4	设备节点写数据	/** * @brief 向设备节点写数据 * @param[in] fd: 设备描述符 * @param[in] buf:写缓存区 * @param[in] len:缓存区长度 * @return 成功返回已写字节长度；失败返回错误码 */ int hal_device_write(int fd, unsigned char *buf, int len);
5	设置设备节点参数	/** * @brief 设置设备参数 * @param[in] fd: 设备描述 * @ param[in] dev_type 设备节点类型 * @ param[in] devattr 设备节点的配置参数 * @return 成功返回 0，失败返回非 0 值 */ int hal_device_set(int fd, int dev_type, void *devattr);

A.11 载波模块接口定义

载波模块接口定义详情见表 A.11。

表 A.11 载波模块接口定义详情表

序号	接口名称	接口定义

1	打开设备节点	/** *@brief 用于配置 int flags 的宏定义,不能同时出现但是必须有一种出现 O_RDWR： 只读模式 O_WRONLY: 只写模式 O_RDWR: 可读可写模式 * @brief 打开设备节点 * @param[in] dev_name: 设备节点名称 * @param[in] flags:打开配置标志，有只读；只写；读写模式 * @return 成功返回设备描述符；失败返回错误码 */ int hal_device_open(char * dev_name,int flags);
2	关闭设备节点	/** * @brief 关闭设备节点 * @param[in] fd:设备描述符 * @return:成功返回 0；失败返回错误码 */ int hal_device_close(int fd);
3	设备节点读数据	/** * @brief 读设备数据 * @param[in] fd: 设备描述符 * @param[out] buf :读缓存区 * @param[in] len: 缓存区长度 * @return 成功返回已读取字节长度；失败返回错误码 */ int hal_device_read(int fd, unsigned char *buf, int len);
4	设备节点写数据	/** * @brief 向设备节点写数据 * @param[in] fd: 设备描述符 * @param[in] buf:写缓存区 * @param[in] len:缓存区长度 * @return 成功返回已写字节长度；失败返回错误码 */ int hal_device_write(int fd, unsigned char *buf, int len);
5	设置设备节点参数	/** * @brief 设置设备参数 * @param[in] fd: 设备描述 * @ param[in] dev_type 设备节点类型 * @ param[in] devattr 设备节点的配置参数 * @return 成功返回 0，失败返回非 0 值 */ int hal_device_set(int fd, int dev_type, void *devattr);

A.12 4G 模块接口定义

4G 模块接口定义详情见表 A.12。

表 A.12 4G 模块接口定义详情表

序号	接口名称	接口定义

1	打开设备节点	/** *@brief 用于配置 int flags 的宏定义,不能同时出现但是必须有一种出现 O_RDWR： 只读模式 O_WRONLY: 只写模式 O_RDWR: 可读可写模式 * @brief 打开设备节点 * @param[in] dev_name: 设备节点名称 * @param[in] flags:打开配置标志，有只读；只写；读写模式 * @return 成功返回设备描述符；失败返回错误码 */ int hal_device_open(char * dev_name,int flags);
2	关闭设备节点	/** * @brief 关闭设备节点 * @param[in] fd:设备描述符 * @return:成功返回 0；失败返回错误码 */ int hal_device_close(int fd);
3	设备节点读数据	/** * @brief 读设备数据 * @param[in] fd: 设备描述符 * @param[out] buf :读缓存区 * @param[in] len: 缓存区长度 * @return 成功返回已读取字节长度；失败返回错误码 */ int hal_device_read(int fd, unsigned char *buf, int len);
4	设备节点写数据	/** * @brief 向设备节点写数据 * @param[in] fd: 设备描述符 * @param[in] buf:写缓存区 * @param[in] len:缓存区长度 * @return 成功返回已写字节长度；失败返回错误码 */ int hal_device_write(int fd, unsigned char *buf, int len);
5	设置设备节点参数	/** * @brief 设置设备参数 * @param[in] fd: 设备描述 * @ param[in] dev_type 设备节点类型 * @ param[in] devattr 设备节点的配置参数 * @return 成功返回 0，失败返回非 0 值 */ int hal_device_set(int fd, int dev_type, void *devattr);

A.13 蓝牙模块接口定义

蓝牙模块接口定义详情见表 A.13。

表 A.13 蓝牙模块接口定义详情表

序号	接口名称	接口定义

1	打开设备节点	/** *@brief 用于配置 int flags 的宏定义,不能同时出现但是必须有一种出现 O_RDWR： 只读模式 O_WRONLY: 只写模式 O_RDWR: 可读可写模式 * @brief 打开设备节点 * @param[in] dev_name: 设备节点名称 * @param[in] flags:打开配置标志，有只读；只写；读写模式 * @return 成功返回设备描述符；失败返回错误码 */ int hal_device_open(char * dev_name,int flags);
2	关闭设备节点	/** * @brief 关闭设备节点 * @param[in] fd:设备描述符 * @return:成功返回 0；失败返回错误码 */ int hal_device_close(int fd);
3	设备节点读数据	/** * @brief 读设备数据 * @param[in] fd: 设备描述符 * @param[out] buf :读缓存区 * @param[in] len: 缓存区长度 * @return 成功返回已读取字节长度；失败返回错误码 */ int hal_device_read(int fd, unsigned char *buf, int len);
4	设备节点写数据	/** * @brief 向设备节点写数据 * @param[in] fd: 设备描述符 * @param[in] buf:写缓存区 * @param[in] len:缓存区长度 * @return 成功返回已写字节长度；失败返回错误码 */ int hal_device_write(int fd, unsigned char *buf, int len);
5	设置设备节点参数	/** * @brief 设置设备参数 * @param[in] fd: 设备描述 * @ param[in] dev_type 设备节点类型 * @ param[in] devattr 设备节点的配置参数 * @return 成功返回 0，失败返回非 0 值 */ int hal_device_set(int fd, int dev_type, void *devattr);

A.14 LED 模块接口定义

LED 模块接口定义详情见表 A.14。

表 A.14 LED 模块接口定义详情表

序号	接口名称	接口定义

1	打开设备节点	/** *@brief 用于配置 int flags 的宏定义,不能同时出现但是必须有一种出现 O_RDWR： 只读模式 O_WRONLY: 只写模式 O_RDWR: 可读可写模式 * @brief 打开设备节点 * @param[in] dev_name: 设备节点名称 * @param[in] flags:打开配置标志，有只读；只写；读写模式 * @return 成功返回设备描述符；失败返回错误码 */ int hal_device_open(char * dev_name,int flags);
2	关闭设备节点	/** * @brief 关闭设备节点 * @param[in] fd:设备描述符 * @return:成功返回 0；失败返回错误码 */ int hal_device_close(int fd);
3	设备节点读数据	/** * @brief 读设备数据 * @param[in] fd: 设备描述符 * @param[out] buf :读缓存区 * @param[in] len: 缓存区长度 * @return 成功返回已读取字节长度；失败返回错误码 */ int hal_device_read(int fd, unsigned char *buf, int len);
4	设备节点写数据	/** * @brief 向设备节点写数据 * @param[in] fd: 设备描述符 * @param[in] buf:写缓存区 * @param[in] len:缓存区长度 * @return 成功返回已写字节长度；失败返回错误码 */ int hal_device_write(int fd, unsigned char *buf, int len);
5	设置设备节点参数	/** * @brief 设置设备参数 * @param[in] fd: 设备描述 * @ param[in] dev_type 设备节点类型 * @ param[in] devattr 设备节点的配置参数 * @return 成功返回 0，失败返回非 0 值 */ int hal_device_set(int fd, int dev_type, void *devattr);

A.15 WDT 模块接口定义

WDT 模块接口定义详情见表 A.15。

表 A.15 WDT 模块接口定义详情表

序号	接口名称	接口定义

1	打开设备节点	/** *@brief 用于配置 int flags 的宏定义,不能同时出现但是必须有一种出现 O_RDWR： 只读模式 O_WRONLY: 只写模式 O_RDWR: 可读可写模式 * @brief 打开设备节点 * @param[in] dev_name: 设备节点名称 * @param[in] flags:打开配置标志，有只读；只写；读写模式 * @return 成功返回设备描述符；失败返回错误码 */ int hal_device_open(char * dev_name,int flags);
2	关闭设备节点	/** * @brief 关闭设备节点 * @param[in] fd:设备描述符 * @return:成功返回 0；失败返回错误码 */ int hal_device_close(int fd);
3	设备节点读数据	/** * @brief 读设备数据 * @param[in] fd: 设备描述符 * @param[out] buf :读缓存区 * @param[in] len: 缓存区长度 * @return 成功返回已读取字节长度；失败返回错误码 */ int hal_device_read(int fd, unsigned char *buf, int len);
4	设备节点写数据	/** * @brief 向设备节点写数据 * @param[in] fd: 设备描述符 * @param[in] buf:写缓存区 * @param[in] len:缓存区长度 * @return 成功返回已写字节长度；失败返回错误码 */ int hal_device_write(int fd, unsigned char *buf, int len);
5	设置设备节点参数	/** * @brief 设置设备参数 * @param[in] fd: 设备描述 * @ param[in] dev_type 设备节点类型 * @ param[in] devattr 设备节点的配置参数 * @return 成功返回 0，失败返回非 0 值 */ int hal_device_set(int fd, int dev_type, void *devattr);

附 录 B
（规范性附录）
接口错误码定义

B.1 接口错误码定义

接口错误码定义见表 B.1。

表 B.1 接口错误码定义

名称	错误码值	错误码定义
接口函数错误码定义	0	成功
	-1	操作不允许
	-2	没有这样的文件或目录
	-3	没有这样的过程
	-4	系统调用被中断
	-5	I/O 错误
	-6	没有这样的设备或地址
	-7	参数列表太长
	-8	执行格式错误
	-9	坏的文件描述符
	-10	没有子进程
	-11	资源暂时不可用
	-12	内存溢出
	-13	拒绝许可
	-14	错误的地址
	-15	块设备请求
	-16	设备或资源忙
	-17	文件存在
	-18	无效的交叉链接
	-19	设备不存在
	-20	不是一个目录
	-21	是一个目录
	-22	无效的参数
	-23	打开太多的文件系统
	-24	打开的文件过多
	-25	不是 tty 设备
	-26	文本文件忙
	-27	文件太大
	-28	设备上没有空间
	-29	非法移位
	-30	只读文件系统

	-31	太多的链接
	-32	管道破裂
	-33	数值结果超出范围
	-34	数值结果不具代表性
	-35	资源死锁错误