旷明 XOS 驱动接口使用说明

部门	
文档编号	
版 本 号	V1.1.0
作　者	

前言

概述

本文主要描述了驱动接口开发组件，仅供参考

产品版本

芯片名称	内核版本
mc331x	linux-4.9.138
qm10xd	linux-5.10.y
qm10xh	linux-5.10.y
qm10xv	linux-4.9

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.1.0		2025-09-04	

目录

一、系统概述

Sysutils 是基于 sysfs 封装的一套用户态接口，包括外设接口和系统功能接口，方便应用层对外设和系统的控制，简化了应用开发难度，方便客户基于这些硬件接口进行应用开发。

二、GPIO

1) 概述

提供 GPIO 基本的用户空间接口。

2) API 参考

2.2.1 qm_gpio_export

【描述】将指定的 GPIO 管脚导出到用户空间，使其可以通过文件系统进行访问。

【语法】int qm_gpio_export(uint32_t gpio)

【参数】

参数名称	描述	输入/输出
uint32_t gpio	GPIO 管脚编号	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】需要具有合适的权限来导出 GPIO 管脚。

【注意】需要查看文件系统中是否有/sys/class/gpio 节点，如果没有该节点，就需要在编译内核时勾选 Device Drivers-> GPIO Support ->/sys/class/gpio/… (sysfs interface)，对应的 CONFIG 名字为 GPIO_SYSFS；GPIO 管脚编号需要在不被占用的状态下；GPIO 管脚编号必须未被导出过，否则导出操作会失败。

【举例】无

2.2.2 qm_gpio_unexport

【描述】取消导出指定的 GPIO 管脚，将其从用户空间中移除。

【语法】int qm_gpio_unexport(uint32_t_gpio)

【参数】

参数名称	描述	输入/输出
uint32_t gpio	GPIO 管脚编号	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】需要在此之前已经通过 qm_gpio_export 成功导出 GPIO 管脚。

【注意】取消导出后，GPIO 将无法再通过用户空间访问。

【举例】无

2.2.3 qm_gpio_set_direction

【描述】设置指定 GPIO 管脚的方向为输入或输出。

【语法】int qm_gpio_set_direction(uint32_t gpio, bool input)

【参数】

参数名称	描述	输入/输出
uint32_t gpio	GPIO 管脚编号	输入
bool input	true:in(输入) false:out(输出)	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】需要确保 GPIO 管脚已导出。

【注意】在设置为输出之前，必须确保没有其他冲突或锁定该 GPIO 的进程。

【举例】无

2.2.4 qm_gpio_get_direction

【描述】获取 GPIO 管脚的方向信息。

【语法】int qm_gpio_get_direction(uint32_t gpio)

【参数】

参数名称	描述	输入/输出
uint32_t gpio	GPIO 管脚编号	输入

【返回值】

返回值	描述
0 或 1	0:out(输出) 1:in(输入)
负	失败

【需求】GPIO 管脚必须已成功导出。

【注意】返回值为 0 表示该 GPIO 设置为输出，为 1 表示输入。

【举例】无

2.2.5 qm_gpio_export_direction

【描述】GPIO 初始化，导出的同时指定 GPIO 的方向。

【语法】int qm_gpio_export_direction(uint32_t gpio, bool input)

【参数】

参数名称	描述	输入/输出
uint32_t gpio	GPIO 管脚编号	输入

bool input	true:in(输入) false:out(输出)	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】必须具有权限来导出并设置 GPIO 的方向。

【注意】此操作会同时导出 GPIO 和设置其方向，避免分别调用 qm_gpio_export() 和 qm_gpio_set_direction()。

【举例】 qm_gpio_export_direction(33,out)

2.2.6 qm_gpio_set_value

【描述】设置 GPIO 管脚的值。

【语法】int qm_gpio_set_value(uint32_t gpio, int value)

【参数】

参数名称	描述	输入/输出
uint32_t gpio	GPIO 管脚编号	输入
int value	0 表示低电平，非零表示高电平	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】GPIO 必须已导出并设置为输出方向。

【注意】如果 GPIO 设置为输入，不能设置其电平值，操作会失败。

【举例】qm_gpio_set_value(33, 1)

2.2.7 qm_gpio_get_value

【描述】获取 GPIO 管脚的值。

【语法】int qm_gpio_get_value(uint32_t gpio)

【参数】

参数名称	描述	输入/输出
uint32_t gpio	GPIO 管脚编号	输入

【返回值】

返回值	描述
0	低电平
1	高电平
非 0	失败

【需求】GPIO 必须已导出并设置为有效方向。

【注意】对于输入 GPIO，可以直接读取电平值;对于输出 GPIO，获取的是当前输出的电平状态。

【举例】qm_gpio_get_value(33)

三、ADC

1) 概述

提供基本的 ADC 用户空间接口。

2) API 参考

3.2.1 qm_adc_get_devnum

【描述】通过设备名获取指定 ADC 设备的设备编号。

【语法】int qm_adc_get_devnum(const char *device_name)

【参数】

参数名称	描述	输入/输出

const char *device_name	表示 ADC 设备名称的字符串	输入

【返回值】

返回值	描述
非负整数	设备编号
负数	失败

【需求】提供有效的 ADC 设备名称；确保设备已被正确识别和注册。

【注意】需要保证在 saradc 初始化之后；设备名称应与系统中的实际名称匹配。

【举例】无

3.2.2 qm_adc_get_raw_value

【描述】获取指定 ADC 通道的采样值。

【语法】int qm_adc_get_raw_value(uint32_t dev_num, uint32_t channel)

【参数】

参数名称	描述	输入/输出
uint32_t dev_num	设备编号	输入
uint32_t channel	该设备所使用的 IIO 通道	输入

【返回值】

返回值	描述
非负整数	转换值
负数	失败

【需求】dev_num 必须是有效的 ADC 设备编号；channel 必须是有效的通道编号。

【注意】必须确保 ADC 设备已初始化，并且目标通道处于可用状态；如果通道编号超出设备支持的范围，函数可能会返回错误。

【举例】qm_adc_get_raw_value(0,0)

3.2.3 qm_adc_get_value

【描述】获取指定 ADC 通道经缩放因子处理后的采样值。

【语法】int qm_adc_get_raw_value(uint32_t dev_num, uint32_t channel)

【参数】

参数名称	描述	输入/输出
uint32_t dev_num	设备编号	输入
uint32_t channel	该设备所使用的 IIO 通道	输入

【返回值】

返回值	描述
非负整数	转换值
负数	失败

【需求】dev_num 必须是有效的 ADC 设备编号；channel 必须是有效的通道编号。

【注意】必须确保 ADC 设备已初始化，并且目标通道处于可用状态；如果通道编号超出设备支持的范围，函数可能会返回错误。

【举例】qm_adc_get_value(0,0)

四、EVENT

1) 概述

按照内核 input event 标准方式，监听按键等事件

2) API 参考

4.2.1 qm_event_register

【描述】注册一个设备路径到事件系统，开始监听指定设备的事件。

【语法】int qm_event_register(char *dev_path)

【参数】

参数名称	描述	输入/输出

char *dev_path	设备的路径，表示你希望监听事件的设备文件	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】设备路径必须有效且设备已连接。

【注意】在注册设备之前，请确保设备已存在并处于可用状态；注册多个设备时，每个设备都需要单独调用此函数。

【举例】无

4.2.2 qm_event_unregister

【描述】取消注册一个设备路径，停止监听该设备的事件

【语法】int qm_event_unregister(char *dev_path)

【参数】

参数名称	描述	输入/输出
char *dev_path	设备的路径，表示你希望停止监听的设备文件	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】设备路径必须已通过 qm_event_register() 成功注册。

【注意】确保设备已经注册，否则调用此函数将无效。

【举例】无

4.2.3 qm_event_listen_start

【描述】启动事件监听系统，并注册一个事件处理函数。此函数将启动事件循环，监听注册设备的事件并触发处理函数。

【语法】int qm_event_listen_start(event_handler_t handler)

【参数】

参数名称	描述	输入/输出
event_handler_t handler	事件处理函数的指针。当监听到设备事件时，将调用此函数来处理事件	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】需要先注册设备后再调用此函数; 必须提供有效的事件处理函数

【注意】不支持重复 qm_event_listen_start；qm_event_listen_start 后不支持再注册监听，但是可以注销监听后调用 qm_event_listen_start，触发事件回调函数接收数据；调用此函数前，需确保所有目标设备已通过 qm_event_register() 注册。

【举例】无

4.2.4 qm_event_listen_stop

【描述】停止事件监听系统，结束事件循环。

【语法】int qm_event_listen_stop(void)

【参数】无

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】调用前必须已启动事件监听系统。

【注意】在停止事件监听之前，所有相关的事件处理和资源管理应当完成；停止事件监听后，不会再接收任何事件，需重新启动才能恢复监听。

【举例】无

五、PWM

1) 概述

提供基本的 pwm 用户空间接口。

2) API 参考

5.2.1 qm_pwm_export

【描述】将指定的 PWM（脉宽调制）设备导出，使其可以被用户空间访问和控制。

【语法】int qm_pwm_export(uint32_t pwm,uint32_t num)

【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm 设备号	输入
uint32_t num	pwm 信号编号	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】设备号和信号编号必须有效且存在。

【注意】需要查看文件系统中是否有/sys/class/pwm 节点，如果没有该节点，就需要在编译内核时勾选 Device Drivers-> Pulse-Width Modulation (PWM) Support，对应的 CONFIG 名字为 PWM。

【举例】无

5.2.2 qm_pwm_unexport

【描述】取消导出指定的 PWM 设备，停止用户空间对其控制。

【语法】int qm_pwm_unexport(uint32_t pwm,uint32_t num)

【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm 设备号	输人
uint32_t num	pwm 信号编号	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】设备号和信号编号必须已通过 qm_pwm_export 成功导出。

【注意】取消导出后，用户空间将无法再访问该 PWM 设备。

【举例】无

5.2.3 qm_pwm_set_period

【描述】设置指定 PWM 设备的周期时间，以纳秒为单位。

【语法】int qm_pwm_set_period(uint32_t pwm,uint32_t period,uint32_t num)

【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm 设备号	输人
uint32_t period	周期时长（纳秒）	输入
uint32_t num	pwm 信号编号	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】pwm 设备必须已导出并处于可操作状态。

【注意】周期时长不超过 10 的 9 次方。

【举例】无

5.2.4 qm_pwm_get_period

【描述】获取指定 PWM 设备的当前周期时间。

【语法】int qm_pwm_get_period(uint32_t pwm,uint32_t num)

【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm 设备号	输人
uint32_t num	pwm 信号编号	输入

【返回值】

返回值	描述
非负整数	周期时长（纳秒）
负数	失败

【需求】pwm 设备必须已导出。

【注意】确保设备在获取时处于工作状态。

【举例】无

5.2.5 qm_pwm_set_duty

【描述】设置指定 PWM 设备的占空比。

【语法】int qm_pwm_set_duty(uint32_t pwm,uint32_t duty,uint32_t num)

【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm 设备号	输人

uint32_t duty	占空比时长（纳秒）	输入
uint32_t num	pwm 信号编号	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】pwm 设备必须已导出；duty 时间不能超过当前设置的周期时间。

【注意】占空比时长不超过 10 的 9 次方，且不超过当前设置的周期时长。

【举例】无

5.2.6 qm_pwm_get_duty

【描述】获取指定 PWM 设备的当前占空比时间。

【语法】int qm_pwm_get_duty(uint32_t pwm,uint32_ num)

【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm 设备号	输人
uint32_t num	pwm 信号编号	输入

【返回值】

返回值	描述
非负整数	占空比时长（纳秒）
负数	失败

【需求】pwm 设备必须已导出并处于工作状态

【注意】确保设备正常运行，才能成功获取占空比

【举例】无

【相关主题】qm_pwm_set_duty

5.2.7 qm_pwm_set_polarity

【描述】设置指定 PWM 设备的极性。

【语法】int qm_pwm_set_polarity(uint32_t pwm, enum pwm_polarity polarity,uint32_t num)

【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm 设备号	输人
enum pwm_polarity polarity	极性值	输入
uint32_t num	pwm 信号编号	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】pwm 设备必须已导出。

【注意】需要确保设置的极性与设备的实际应用相匹配。

【举例】无

5.2.8 qm_pwm_get_polarity

【描述】获取指定 PWM 设备的当前极性。

【语法】int qm_pwm_get_polarity(uint32_t pwm,uint32_t num)

【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm 设备号	输人
uint32_t num	pwm 信号编号	输入

【返回值】

返回值	描述
0 或 1	0：正极性， 1：负极性
负	失败

【需求】pwm 设备必须已导出并运行。

【注意】确保设备运行正常才能成功获取极性。

【举例】无

5.2.9 qm_pwm_set_enable

【描述】启用或禁用指定的 PWM 设备。

【语法】int qm_pwm_set_enable(uint32_t pwm, bool enabled,uint32_t num)

【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm 设备号	输人
bool enabled	是否启用 PWM 输出，true 为启用，false 为禁用	输入
uint32_t num	pwm 信号编号	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】pwm 设备必须已导出。

【注意】确保设置后，设备的启用状态符合应用需求。

【举例】无

5.2.10 qm_pwm_get_enable

【描述】获取指定 PWM 设备的启用状态。

【语法】int qm_pwm_get_enable(uint32_t pwm,uint32_t num)

【参数】

参数名称	描述	输入/输出
uint32_t pwm	pwm 设备号	输人
uint32_t num	pwm 信号编号	输入

【返回值】

返回值	描述
0 或 1	0：禁用， 1：启用
负	失败

【需求】pwm 设备必须已导出。

【注意】确保设备处于工作状态才能成功获取启用状态。

【举例】无

5.2.11 qm_pwm_init

【描述】初始化指定的 PWM 设备，包括周期、占空比和极性的设置。

【语法】int qm_pwm_init(uint32_t pwm, uint32_t period, uint32_t duty, enum pwm_polarity polarity,uint32_t num)

【参数】

参数名称	描述	输入/输出
uint32_t pwm	PWM 设备号	输人
uint32_t period	周期时长（纳秒）	输入
uint32_t duty	占空比时长（纳秒）	输入
enum pwm_polarity polarity	极性设置	输入
uint32_t num	PWM 信号编号	输入

【返回值】

返回值	描述

0	成功
非 0	失败

【需求】pwm 设备必须已导出。

【注意】无

【举例】qm_pwm_init(0, 1000000, 500000, PWM_POLARITY_NORMAL, 9);

5.2.12 qm_pwm_deinit

【描述】反初始化指定的 PWM 设备，停止其工作并释放相关资源。

【语法】int qm_pwm_deinit(uint32_t pwm,uint32_t num)

【参数】

参数名称	描述	输入/输出
uint32_t pwm	PWM 设备编号	输入
uint32_t num	PWM 信号编号	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】pwm 设备必须已导出并工作。

【注意】反初始化后，设备将停止工作，所有相关设置将被清除。

【举例】无

3) 数据类型

5.3.1 enum pwm_polarity

【说明】 pwm 极性

【定义】

```
enum pwm_polarity {
PWM_POLARITY_NORMAL,
PWM_POLARITY_INVERSED,
};
```

【成员】

成员名称	描述
PWM_POLARITY_NORMAL	正极性
PWM_POLARITY_INVERSED	负极性

六、TIME

1) 概述

提供硬件时间和系统时间的用户空间接口。

2) API 参考

6.2.1 qm_system_get_time

【描述】获取 rtc 时间，并将结果存储到提供的 struct tm 结构体中。

【语法】int qm_system_get_time(struct tm *time)

【参数】

参数名称	描述	输入/输出
struct tm *time	指向存储当前系统时间的 tm 结构体指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】需要确保传入的参数有效。

【注意】需要确保/dev/rtc 节点开启，如果没有开启，就需要在编译内核时勾选 Device Drivers-> Real Time Clock，对应的 CONFIG 名字为 RTC_CLASS。

【举例】无

6.2.2 qm_system_set_time

【描述】设置系统时间，将提供的 tm 结构体中的时间值应用于系统中。

【语法】int qm_system_set_time(struct tm *time)

【参数】

参数名称	描述	输入/输出
struct tm *time	指向要设置的 tm 结构体指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】需要确保 struct tm 结构体中的时间有效且格式正确。

【注意】需要确保/dev/rtc 节点开启，如果没有开启，就需要在编译内核时勾选 Device Drivers-> Real Time Clock ，对应的 CONFIG 名字为 RTC_CLASS；需要输入有效的时间。

【举例】无

6.2.3 qm_system_set_alarm

【描述】获取当前系统设定的报警时间，并将结果存储在 struct tm 结构体中。

【语法】qm_system_set_alarm(struct tm *time)

【参数】

参数名称	描述	输入/输出
struct tm time	指向存储报警时间的 tm 结构体指针	输入

【返回值】

返回值	描述

0	成功
非 0	失败

【需求】需要确保 struct tm 结构体指针有效。

【注意】alarm 中断的触发时间只能是 24 小时内的一个时刻，所以只有时、分、秒的部分是有效的，参数 time 的年、月、日部分会被忽略。

【举例】无

6.2.4 qm_system_get_alarm

【描述】设置系统报警时间，将提供的 tm 结构体中的时间值设定为报警时间。

【语法】int qm_system_get_alarm(struct tm *time)

【参数】

参数名称	描述	输入/输出
struct tm *time	指向要设置的报警时间的 tm 结构体指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】需要确保 struct tm 结构体中提供的时间值有效。

【注意】需要在 qm_system_set_alarm 成功后调用 qm_system_get_alarm。

【举例】无

6.2.5 qm_system_enable_alarm

【描述】启用系统报警功能，使系统报警机制开始工作。

【语法】int qm_system_enable_alarm(void)

【参数】无

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】需要在设置报警时间后启用报警功能。

【注意】需要在 qm_system_set_alarm 成功后调用 qm_system_enable_alarm，不支持 enable 后重新设置 alarm 的触发时间。

【举例】无

6.2.6 qm_system_disable_alarm

【描述】禁用系统报警功能，停止任何已设定的报警。

【语法】int qm_system_disable_alarm(void)

【参数】无

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】在不需要报警功能时，可以通过此 API 禁用报警。

【注意】禁用报警后，已设定的报警时间将失效，直到重新启用报警。

【举例】无

6.2.7 qm_system_wait_alarm

【描述】阻塞进程，直到设定的报警时间到达或达到指定的超时时间。

【语法】int qm_system_wait_alarm(uint32_t wait_seconds)

【参数】

参数名称	描述	输入/输出

uint32_t wait_seconds	超时时间，单位为毫秒。如果为 0，表示无限等待	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】必须设置报警时间并启用报警功能，才能有效等待报警。

【注意】阻塞当前线程，被 qm_system_enable_alarm 调用。

【举例】无

七、LED

1) 概述

提供 LED 的用户空间接口。

2) API 参考

7.2.1 qm_led_set_mode

【描述】设置指定 LED 设备的工作模式，包括是否闪烁和亮度设置

【语法】int qm_led_set_mode(char *dev_path, bool blink, uint32_t brightness)

【参数】

参数名称	描述	输入/输出
char *dev_path	LED 设备的设备路径，指向设备文件的字符串	输入
bool blink	设置 LED 是否闪烁，true 为闪烁，false 为常亮	输入
uint32_t brightness	LED 亮度值，范围通常为 0（关闭）到最大亮度值	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】LED 设备路径 (dev_path) 必须有效且指向正确的设备；亮度值 (brightness) 应在设备允许的范围内。

【注意】需要保证/sys/class/leds 设备节点存在，如果不存在，就需要在编译内核时勾选 Device Drivers-> LED Support ->LED Class Support，对应的 CONFIG 名字为 LEDS_CLASS；当设备不支持闪烁功能时，设置 blink 为 true 可能无效。

【举例】无

八、WATCHDOG

1) 概述

提供看门狗的用户空间接口。

2) API 参考

8.2.1 qm_watchdog_start

【描述】启动看门狗定时器，并设置超时时间

【语法】int qm_watchdog_start(int timeval)

【参数】

参数名称	描述	输入/输出
int timeval	超时时间，单位为秒。当超时未被刷新时，系统将重启	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】看门狗必须在设备支持的情况下使用。

【注意】在超时时间内，需定期刷新看门狗，以防止系统重启。

【举例】无

8.2.2 qm_watchdog_refresh

【描述】喂狗，刷新看门狗定时器，防止系统重启。

【语法】int qm_watchdog_refresh(void)

【参数】无

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】看门狗必须已启动才能刷新。

【注意】刷新应在超时时间内定期调用，以保持系统稳定。

【举例】无

8.2.3 qm_watchdog_stop

【描述】停止正在运行的看门狗定时器。

【语法】int qm_watchdog_stop(void)

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】看门狗必须已启动才能停止。

【注意】停止看门狗后，系统不再受到监控。

【举例】无

九、SYSTEM

1) 概述

提供系统操作的用户空间接口，包括设备重启，休眠，关机，CHIPID 以及 SN 号的获取。

2) API 参考

9.2.1 qm_chip_id_get

【描述】获取芯片的唯一标识符，并将其存储在提供的字符数组中。

【语法】int qm_chip_id_get(char *chipid)

【参数】

参数名称	描述	输入/输出
char *chipid	指向字符数组的指针，用于存储芯片 ID	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】chipid 指针必须指向足够大的字符数组以存储芯片 ID。

【注意】确保在调用此函数前，chipid 数组已正确分配内存。

【举例】无

9.2.2 qm_vendor_write

【描述】向 vendor 写数据

【语法】int qm_vendor_write(int vendor_id, const char *data, int size)

【参数】

参数名称	描述	输入/输出
int vendor_id	编号	输入

const char *data	待写入数据指针	输入
int size	待写入数据大小	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】确保 data 指向有效的数据并且 size 正确。

【注意】必须保证/dev/vendor_storage 存在；size 的大小建议在 1024 字节内。

【举例】无

9.2.3 qm_vendor_read

【描述】读取 vendor 的数据并存储在提供的缓冲区中。

【语法】int qm_vendor_write(int vendor_id, const char *data, int size)

【参数】

参数名称	描述	输入/输出
int vendor_id	编号	输入
char *data	指向缓冲区的指针，用于存储读取的数据	输出
int size	读取数据大小	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】data 缓冲区必须足够大以存储读取的数据。

【注意】必须保证/dev/vendor_storage 存在。

【举例】无

9.2.4 qm_system_reboot

【描述】系统重启。

【语法】int qm_system_reboot(void)

【参数】无

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】必须具有适当的权限才能重启系统。

【注意】在调用此函数之前请确保已保存所有文件。

【举例】无

9.2.5 qm_system_shutdown

【描述】关闭系统

【语法】int qm_system_shutdown(void)

【参数】无

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】必须具有适当的权限才能关闭系统。

【注意】在调用此函数之前请确保已保存所有文件。

【举例】无

9.2.6 qm_system_suspend

【描述】系统休眠。

【语法】int qm_system_suspend(SUSPEND_TYPE type)

【参数】

参数名称	描述	输入/输出
SUSPEND_TYPE type	休眠类型，枚举值	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】休眠操作需要适当的权限。

【注意】FREEZE 与 MEM 的休眠方式需要查看系统是否支持。

【举例】无

9.2.7 qm_parse_cmdline

【描述】该函数用于从程序启动时的命令行参数中，查找指定的字符串，并将对应的参数字符串提取出来。

【语法】int qm_parse_cmdline(const char *key, char* pdata)

【参数】

参数名称	描述	输入/输出
key	在命令行中查找的关键字	输入
pdata	存储查找到的参数字符串	输入

【返回值】

返回值	描述
0	成功

非 0	失败或数据为空

【需求】调用此函数时，必须确保程序启动时已经传入正确格式的命令行参数，并且 pdata 缓冲区足够大。

【注意】pdata 缓冲区长度需足够，否则可能截断数据；key 参数不能为空。

【举例】无

9.2.8 qm_vendor_prop_get

【描述】该函数根据传入的参数__request，调用 qm_parse_cmdline 函数分别获取对应的属性内容。

【语法】static int qm_vendor_prop_get(unsigned long int __request,void *pdata)

【参数】

参数名称	描述	输入/输出
unsigned long int __request	用于指定所需获取的属性类型	输入
void *pdata	指向存储属性数据的缓冲区指针，并将结果写入缓冲区	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】系统中必须支持 qm_parse_cmdline 函数；pdata 参数必须有足够空间存储返回的数据。

【注意】当__request 的值不匹配已定义的请求类型时，会返回-1；确保传入的 pdata 指针已正确分配内存，已避免内存写入错误。

【举例】无

9.2.9 qm_hardware_model_get

【描述】调用 qm_vendor_prop_get 获取硬件型号信息，将结果写入提供的缓冲区中。

【语法】int qm_hardware_model_get(void *pdata)

【参数】

参数名称	描述	输入/输出
void *pdata	指向存储硬件型号字符串的缓冲区	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】设备的启动参数中必须包含与硬件型号相关的配置信息

【注意】无

【举例】无

9.2.10 qm_product_model_get

【描述】调用 qm_vendor_prop_get 获取产品型号信息，将解析后的数据保存到提供的缓冲区中。

【语法】int qm_product_model_get(void *pdata)

【参数】

参数名称	描述	输入/输出
void *pdata	指向存放产品型号数据的缓冲区	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】设备启动时必须提供产品型号的相关配置信息。

【注意】无

【举例】无

9.2.11 qm_serial_id_get

【描述】调用 qm_vendor_prop_get 获取设备的序列号信息，并把数据写入提供的缓冲区中。

【语法】int qm_serial_id_get(void *pdata)

【参数】

参数名称	描述	输入/输出
void *pdata	指向存储序列号信息的缓冲区	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】系统启动参数中需要包含设备序列号信息。

【注意】无

【举例】无

9.2.12 qm_device_type_get

【描述】调用 qm_vendor_prop_get 获取设备类型信息，结果保存在传入的缓冲区中。

【语法】int qm_device_type_get(void *pdata)

【参数】

参数名称	描述	输入/输出
void *pdata	指向保存设备类型数据的缓冲区	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】设备类型信息需要在启动参数中进行配置。

【注意】无

【举例】无

9.2.13 qm_manufacture_get

【描述】调用 qm_vendor_prop_get 获取制造商信息，将解析结果保存在参数指向的缓冲区中。

【语法】int qm_manufacture_get(void *pdata)

【参数】

参数名称	描述	输入/输出
void *pdata	指向存放制造商信息的缓冲区	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】制造商信息必须在设备启动命令行参数中预先配置好

【注意】无

【举例】无

9.2.14 qm_sdk_version_get

【描述】获取 SDK 版本信息。若预编译宏被定义，则直接将其字符串拷贝到 pdata；否则调用 qm_vendor_prop_get 从命令行参数中解析 SDK 版本信息。

【语法】int qm_sdk_verison_get(void *pdata)

【参数】

参数名称	描述	输入/输出
void *pdata	指向存放 SDK 版本信息的缓冲区	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】如果未定义编译宏，则在系统启动参数中需配置 SDK 版本信息。

【注意】无

【举例】无

3) 数据类型

9.3.1 SUSPEND_TYPE

【说明】 休眠类型

【定义】

```
typedef enum {
SUSPEND_FREEZE = 0,
SUSPEND_MEM,
} SUSPEND_TYPE;
```

【成员】

成员名称	描述
SUSPEND_FREEZE	FREEZE 休眠方式
SUSPEND_MEM	MEM 休眠方式

十、BACKLIGHT

1) 概述

提供 BACKLIGHT 基本的用户空间接口，包括获取亮度，设置亮度等。

2) API 参考

10.2.1 qm_backlight_getbri

【描述】获取背光亮度。

【语法】int qm_backlight_getbri()

【参数】无

【返回值】

返回值	描述
0-100	获取成功，即背光亮度值
-1	失败

【需求】无

【注意】用户可见亮度值为 0-100。

【举例】无

10.2.2 qm_backlight_setbri

【描述】设置背光亮度。

【语法】int qm_backlight_setbri(uint32_t brightness)

【参数】

参数名称	描述	输入/输出
brightness	亮度值	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

【注意】用户可修改亮度值为 0-100。

【举例】qm_backlight_setbri(100)

十一、HOTPLUG

1) 概述

提供热插拔基本的用户空间接口。

2) API 参考

11.2.1 qm_usbHotplug

【描述】用于注册一个 USB 热插拔处理回调函数，并启动 USB 热插拔检测功能。当检测到 USB 设备的插入或移除事件时，系统会调用用户提供的回调函数 callback，以便用户可以自定义事件处理逻辑。

【语法】int qm_usbHotplug(HotPlugCallback callback)

【参数】

参数名称	描述	输入/输出
HotPlugCallback callback	用户定义的回调函数，用于处理 USB 设备热插拔事件	输入

【返回值】

返回值	描述
0	成功，USB 热插拔监听已启动
负值	失败

【需求】必须在目标设备支持 USB 热插拔功能的前提下使用此函数；调用者需要提供一个有效的回调函数，确保能够正确处理 USB 插入和移除事件。

【注意】调用此函数前，请确保已正确初始化 USB 子系统，避免因系统未初始化而导致无法正常调用。

【举例】无

11.2.2 qm_hotPlug_stop

【描述】停止热插拔监控系统，终止设备插入和移除事件的监听。

【语法】int qm_hotPlug_stop()

【参数】无

【返回值】

返回值	描述
0	成功
非零	失败

【需求】调用此函数来清理资源并停止监控。

【注意】调用此函数会阻塞当前线程，直到热插拔监控线程完全退出。

【举例】无

十二、MASS_STORAGE

1) 概述

提供 USB 大容量存储模式基本的用户空间接口。

2) API 参考

12.2.1 qm_mass_storage_enable

【描述】该函数用于配置并启用 USB Mass Storage 模式。它通过挂载 configfs，设置 USB 描述符，创建配置文件和目录，并将指定的存储文件路径挂载到 USB Gadget 中，从而使设备以 USB 存储设备的形式对外可见。

【语法】int qm_mass_storage_enable(const char *file_path)

【参数】

参数名称	描述	输入/输出
const char *file_path	被挂载到 USB Mass Storage 的	输入

	文件路径	

【返回值】

返回值	描述
0	成功
-1	失败

【需求】设备必须 Device 模式；内核必须启用 USB Gadget 框架和 Mass Storage 功能；configfs 文件系统必须被正确挂载到 /sys/kernel/config。

【注意】配置完成后，主机端会识别设备为一个 USB 存储设备。根据主机系统的不同，可能需要一些时间完成挂载；调用此函数前可能需要调用 qm_mass_storage_disable 以确保清理旧的 USB Gadget 配置；确保传入的 file_path 是有效的文件或设备路径，并且主机系统具有对该文件的读写权限。

【举例】无

12.2.2 qm_register_pc_connect_event_callback

【描述】该函数用于注册 PC 连接状态变化的事件回调函数。当 USB 设备状态发生变化时，注册的回调函数会被触发，并将当前 PC 连接状态作为参数传递。注册首个回调时，内部会启动监控线程，持续监控 USB 状态文件的变化。

【语法】int qm_register_pc_connect_event_callback(qm_pc_connect_event_callback callback)

【参数】

参数名称	描述	输入/输出
qm_pc_connect_event_callback callback	回调函数指针	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】调用前确保回调函数非空，并且系统具有对 USB 设备状态监控的权限。

【注意】同一回调函数不能重复注册。

【举例】无

12.2.3 qm_unregister_pc_connect_event_callback

【描述】该函数用于注销之前注册的 PC 连接状态变化事件回调函数。注销后，该回调函数将不再收到 USB 设备状态变化的通知。如果注销的是最后一个回调，内部会停止监控线程并释放相关资源。

【语法】int qm_unregister_pc_connect_event_callback(qm_pc_connect_event_callback callback)

【参数】

参数名称	描述	输入/输出
qm_pc_connect_event_callback callback	回调函数指针	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】调用前确保传入的回调函数成功注册且当前处于有效状态。

【注意】在注销过程中，函数首先将回调标记为无效，并等待可能正在执行的回调结束，然后销毁其互斥锁资源；如果注销的是最后一个有效回调，系统将调用 stop_monitor() 停止 USB 监控，并重置 PC 连接状态为未知；多线程环境下，通过读写锁确保回调数组的线程安全操作。

【举例】无

十三、POWCHARGE

1) 概述

提供充电模块基本的用户空间接口，包括充电状态监测，电池电量查询，是否充满等

2) API 参考

13.2.1 qm_start_battery_monitor

【描述】该函数用于启动电池监控线程，用于实时监测电池电量百分比。启动后，系统会定时采样电池电压，并计算出电量百分比。每次更新时会调用用户注册的回调函数，将最新的电池电量（0~100%）通知上层应用。

【语法】int qm_start_battery_monitor(void (*callback)(int));

【参数】

参数名称	描述	输入/输出
callback	回调函数指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】系统须支持 pthread 多线程，且 ADC 设备必须工作正常以提供稳定的电压采样数据。

【注意】启动后监控线程会每 2 秒更新一次电池百分比；使用后可调用 qm_stop_battery_monitor 停止监控，释放线程资源。

【举例】无

13.2.2 qm_startup_battery_percentage

【描述】该函数用于获取系统开机时的电池充电百分比。函数首先尝试读取关机前保存的电池信息文件，如果文件中的数据与当前采样数据差异不大（差距≤15%），则采用关机数据；否则以当前实时数据为准。这样可以避免开机时因采样不稳定而导致电量读数异常。

【语法】int qm_startup_battery_percentage(void)

【参数】无

【返回值】

返回值	描述
0-100	返回电池百分比
负数	失败

【需求】系统需已启用关机回调保存电池数据，并且 ADC 设备正常工作以采集准确的电压数据。

【注意】使用关机数据前会判断数据与实时数据的差异，确保数据准确；若关机保存文件不存在或数据无效，则自动使用当前采样数据。

【举例】 无

13.2.3 qm_get_battery_percentage

【描述】用于获取当前电池的电量百分比，通过读取电池电压来计算电池的电量。

【语法】int qm_get_battery_percentage()

【参数】无

【返回值】

返回值	描述
-1	失败
整数	成功，表示电池电量百分比

【需求】无

【注意】无

【举例】无

13.2.4 qm_is_battery_full

【描述】用于检查电池电压是否已达到最大电压值，判断电池是否充满。当电池电压大于或等于设定的最大电压值时，返回电池已充满的状态。若读取电压失败，则输出错误信息并返回失败。

【语法】bool qm_is_battery_full()

【参数】无

【返回值】

返回值	描述
0	未充满/读取电压失败
1	充满

【需求】 VOLTAGE_MAX 应该定义为电池充满时的最大电压值。

【注意】无

【举例】无

13.2.5 qm_monitor_charging_start

【描述】该函数用于启动充电状态监控线程，监测充电状态的变化。当检测到充电状态（开始或停止充电）变化时，会通过回调函数通知上层应用，方便实时掌握充电情况。

【语法】int qm_monitor_charging_start(void (*callback)(int))

【参数】

参数名称	描述	输入/输出
callback	回调函数指针，函数参数为 int，值为 1 表示开始充电，0 表示停止充电	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】确保系统具备创建线程和读取 GPIO 状态的能力，且硬件配置正常。

【注意】充电监控线程每秒检测一次充电状态。

【举例】无

十四、MOTOR

1) 概述

提供马达模块的用户空间接口。

2) API 参考

14.2.1 qm_motor_manualStop

【描述】用于手动停止指定马达的运行。函数先发送停止命令（FH_SM_STOP）给目标马达，并在命令执行成功后再次发送命令检查马达是否已停止运行（通过 FH_SM_CHECKISTOP）。

【语法】int qm_motor_manualStop(int *idx*)

【参数】

参数名称	描述	输入/输出
Idx	待停止的电机索引号，通过该索引从电机文件描述符数组中选择目标电机。	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】与电机控制相关的 ioctl 接口（如 FH_SM_STOP 和 FH_SM_CHECKISTOP）已正确初始化；文件描述符数组（例如 stmFd）及相关全局变量（如 bRun）均已配置完毕。

【注意】本函数依赖于 ioctl 系统调用，若 ioctl 调用失败（例如设备未连接或命令不支持），会直接返回错误码；返回错误码时，日志中会输出错误信息，调用者应根据实际返回值判断错误原因。

【举例】无

14.2.2 qm_motor_run_control

【描述】该函数用于控制马达的运行。根据传入的参数设置马达旋转的方向、步数和速度，同时自动完成设备的打开、更新以及设备的关闭操作，确保马达能够按照指定参数进行运动控制。

【语法】int qm_motor_run_control(int *idx*, int *direction*, int *step*, int *speed*)

【参数】

参数名称	描述	输入/输出
idx	马达索引号	输入
direction	运行方向，非 0 表示正向运行，0 表示反向运行	输入
step	运行步数，决定马达旋转的步数或距离	输入

speed	电机运行速度	输入

【返回值】

返回值	描述
0	成功

【需求】在调用本函数前，应确保目标设备对应的文件或设备句柄已经正确打开。函数内部会自动调用 test_stm_open()进行设备打开、调用更新函数以及最终调用 test_stm_close()来关闭设备，要求运行环境支持 ioctl 接口等底层硬件接口。

【注意】函数内部会更改全局变量，调用时需确保不会影响其他并发操作；设备的打开、更新和关闭流程是必须的，若某一步失败可能会导致返回失败状态。

【举例】无

14.2.3 qm_motor_control

【描述】该函数用于初始化马达配置，设置马达开关，震动强度，震动时间。

【语法】int qm_motor_control(int index,bool enabled,int intensity,int time)

【参数】

参数名称	描述	输入/输出
int index	PWM 使能	输入
bool enabled	震动开关	输入
int intensity	震动强度，范围 0-100	输入
int time	震动时间，-1 为常震动，其他为定时震动，单位 us	输入

【返回值】

返回值	描述
0	成功

负数	失败

【需求】文件系统中已加载/etc/sysutils.cfg 文件，且包含正确的 CONFIG_MOTOR_EN 和 CONFIG_MOTOR_PWM 配置

【注意】该函数适用双路 PWM 配置(参考原理图)，其他情况可能会失效

【举例】qm_motor_control(9, true, 80, 20000)

十五、WIFI&BT

1) 概述

负责 WIFI 和蓝牙相关的配置和管理，包括 WIFI 连接、断开、状态检测以及网络扫描等。

2) API 参考

15.2.1 qm_wifi_wlan0_down

【描述】该函数用于关闭无线局域网接口 wlan0，通过调用 system() 函数执行系统命令 "ifconfig wlan0 down"，使无线网络接口停止工作。

【语法】int qm_wifi_wlan0_down(void)

【参数】无

【返回值】

返回值	描述
0	成功

【需求】需要在系统中安装 ifconfig 工具，并且调用此函数时具有足够权限（通常需要 root 权限）以执行网络接口的关闭操作。

【注意】该函数调用 system() 执行系统命令，下线无线接口可能会影响系统的网络连接；不同系统中可能使用不同的命令管理网络接口，调用此函数前需确保 ifconfig 命令可用。

【举例】无

15.2.2 qm_wifi_connect_status

【描述】该函数用于检测 WIFI 连接状态。简单来说，它会先检查无线网络接口（wlan0）的状态，确认接口是否处于“UP”状态，并且是否获得了 IP 地址；接着，它从系统路由信息中提取默认网

关地址，利用该地址执行 ping 测试来判断网络是否通畅。如果最后 ping 测试成功，则说明 WIFI 连接正常，函数返回 1，否则返回 0。

【语法】int qm_wifi_connect_status(void)

【参数】无

【返回值】

返回值	描述
1	Wifi 正常连接
0	Wifi 未连接或网络测试失败

【需求】在调用该函数前，需要确保系统已正确配置 WPA Supplicant 和相关无线网卡驱动，同时具备执行网络命令（如 ip addr、ip route、ping 等）的权限。若无线配置存在问题，函数还会尝试复制配置文件并重新执行连接命令。

【注意】函数内部依赖于系统命令（如 system("ip addr show wlan0")、system("ip route")、system(pingCmd)），执行这些命令需要有相应的权限；函数会在/tmp 目录下生成临时文件（如 /ipaddess.txt、/route.txt、/ping.txt），请保证对这些文件有读写权限；网络状态不佳时，函数可能会频繁调用重新连接命令（qm_connection_wifi_cmd），这可能会影响系统性能或网络稳定性。

【举例】无

15.2.3 qm_wifi_status_query

【描述】这个函数主要用于查询 WIFI 的状态。它的做法是构造一个调用外部脚本的命令字符串，并执行这个命令来检测 WIFI 状态。函数会把固定脚本名 "wifistart.sh" 与一个等待时间参数拼接在一起，如果传入参数非空，则使用传入值；否则使用默认值 "5"。最后打印出构造的命令，并通过 system() 函数执行，返回 system() 的执行结果。

【语法】int qm_wifi_status_query(const char *witTims);*

【参数】

参数名称	描述	输入/输出
const char *witTims*	指定 WIFI 状态查询时的等待时间	输入

【返回值】

返回值	描述
0	执行成功
非 0	异常错误

【需求】要求系统中存在名为 "wifistart.sh" 的脚本，并且该脚本能接收一个参数（如等待时间）来执行与 WIFI 状态相关的操作。同时，执行此函数需具备调用系统命令的权限。

【注意】传入参数 witTims 应为字符串，代表希望传递给脚本的参数；若为空，则自动使用默认值 "5"；函数使用 system() 函数执行外部命令，因此需要适当的系统权限，并确保脚本文件存在且有执行权限。

【举例】qm_wifi_status_query("10");

15.2.4 qm_wifi_cfg

【描述】该函数用于配置 WiFi 网络。它会将提供的 WiFi 网络名称（SSID）、密码（PSK）和密钥管理方式（key_mgmt）写入到 wpa_supplicant 的配置文件中，从而为无线网络连接做好准备。函数的主要操作包括：创建存放配置文件的目录、打开并写入配置文件内容、以及调用 WiFi 连接命令 qm_connection_wifi_cmd。

【语法】int qm_wifi_cfg(const char *ssid, const char* passwd, const char *key)*

【参数】

参数名称	描述	输入/输出
const char *ssid*	Wifi 网络名称	输入
const char passwd	Wifi 网络密码	输入
const char *key*	Wifi 密钥管理方式（如 WPA-PSK 等），用于指定认证模式	输入

【返回值】

返回值	描述
1	表示在创建或写入配置文件时发生错误
0	表示配置文件写入成功且连接命令执行正

	常

【需求】需要系统中已安装 wpa_supplicant，并且具备在/data 目录下创建 wifi 目录以及生成文件的权限。同时，还需要保证 qm_connection_wifi_cmd()能够正常调用以启动 WiFi 连接流程。

【注意】函数内部调用了 system()函数执行 shell 命令，确保系统环境支持相关命令；传入的参数必须为合法的字符串（非 NULL），否则对应的配置行不会写入；

【举例】qm_wifi_cfg("MyWifi", "12345678", "WPA-PSK");

15.2.5 qm_get_ssid_psw

【描述】这个函数的作用是从 WiFi 配置文件 "/data/wifi/wpa_supplicant.conf" 中读取当前设置的 WiFi 网络名称（SSID）和密码（PSK），然后把它们分别保存到调用者提供的两个字符数组中。简单地说，就是用来提取配置文件里的 WiFi 名称和密码。

【语法】int qm_get_ssid_psw(char s*sid, char* passwd)

【参数】

参数名称	描述	输入/输出
char s*sid*	WIFI 网络名称	输入
char passwd	WiFi 密码	输入

【返回值】

返回值	描述
0	成功
1	失败

【需求】在调用本函数前，需要确保文件 "/data/wifi/wpa_supplicant.conf" 存在且具有读取权限，且传入的 ssid 和 passwd 缓冲区足够大，可以存放提取到的字符串。

【注意】如果配置文件中没有找到 “ssid=” 或 “psk=” 关键字，相应的缓冲区可能会为空。

【举例】无

15.2.6 qm_wifi_get_ssid_list

【描述】获取周围 WiFi 网络的信息列表。它会先调用外部脚本生成/刷新 SSID 信息文件（/data/SSID.txt），然后打开这个文件，逐行读取数据，提取每一行中包含的 WiFi 信号强度和 SSID 名称，并将这些信息保存到传入的结构体中。最后，它会将收集到的所有 WiFi 网络根据信号强度进行排序，方便后面选择信号较好的网络进行连接。

【语法】int qm_wifi_get_ssid_list(wifi_ssid_list * *ssidList*)

【参数】

参数名称	描述	输入/输出
wifi_ssid_list * *ssidList*	存放 WiFi 网络列表的结构体指针	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】系统中存在"/data/SSID.txt"这个文件，该文件由外部脚本（flash_ssid_one.sh）生成并包含 WiFi 扫描结果；传入的 ssidList 结构体已经正确分配，并且能够存储足够数量的 WiFi 信息。

【注意】在调用过程中，通过 system()函数执行外部脚本“flash_ssid_one.sh”刷新 SSID 信息，如果该脚本执行失败，函数会直接返回错误；函数会忽略超过最大限制（QM_MAX_SSID_NUM）的 WiFi 网络；读入每行数据后，会先判断是否包含“signal:”来获得信号强度，如果没有，再判断是否包含“SSID:”来获取网络名称。

【举例】无

15.2.7 qm_wifi_start_wlan0_scan

【描述】用于启动 wlan0 无线网卡的扫描操作。简单来说，它会先启用 wlan0 网卡，然后删除旧的 SSID 文件，接着利用 iw 命令结合 awk 工具扫描周围的无线网络，并提取出每个网络的信号强度和名称，最后将这些信息保存到/home/SSID.txt 文件中。

【语法】int qm_wifi_start_wlan0_scan(void)

【参数】无

【返回值】

返回值	描述
0	成功

【需求】系统中必须安装并配置好 ifconfig、iw 和 awk 等命令工具；用户需具有执行 system 命令以及读写/home 目录下文件的权限；wlan0 无线网卡及相关驱动必须正常工作。

【注意】扫描命令的输出会写入到/home/SSID.txt 文件，确保/home/SSID.txt 路径存在且可写。

【举例】无

十六、SD

1) 概述

提供一些 TF/SD card 的用户空间接口。

2) API 参考

16.2.1 qm_format_sd

【描述】这个函数用于格式化 SD 卡。它首先检查指定的设备是否存在，如果存在则卸载设备上所有已经挂载的分区，接着将设备格式化为 FAT32 文件系统，并最后将格式化后的设备挂载到/mnt/sdcard 目录。整个流程成功完成则返回 0，否则在任何步骤发生错误时返回-1。

【语法】int qm_format_sd(void)

【参数】无

【返回值】

返回值	描述
0	成功
-1	失败

【需求】调用该函数前，确保系统中已正确设置 device_path 变量，并且 SD 卡已经连接到系统上；系统需要具有执行 umount、mkfs.vfat 以及 mount 等命令的权限。

【注意】如果设备不存在，函数会使用 perror 输出错误信息并返回-1；卸载分区使用的是"umount %s"命令，如果分区没有挂载或卸载失败，会输出警告信息，但格式化操作仍可继续；格式化过程中如果 mkfs 命令执行失败，函数会返回-1；挂载设备到/mnt/sdcard 时，也可能因权限或路径问题导致失败。

【举例】无

16.2.2 qm_get_storage_capacity_by_path

【描述】该函数用于获取指定存储设备路径的存储容量信息，包括总容量，已用容量和剩余容量。函数通过 statfs 系统调用获取对应存储设备的文件系统信息，并通过计算得出所需的容量数据。

【语法】int64_t qm_get_storage_capacity_by_path(const char* path,uint64_t total_capacity,uint64_t used_capacity,uint64_t free_capacity)

【参数】

参数名称	描述	输入/输出
const char* path	存储设备的挂载路径，必须为有效的路径字符串	输入
uint64_t total_capacity	存储设备的总容量	输出
uint64_t used_capacity	存储设备的已用容量	输出
uint64_t free_capacity	存储设备的剩余容量	输出

【返回值】

返回值	描述
0	成功
-1	失败

【需求】传入的 path 参数必须指向一个有效的存储设备挂载路径，且系统支持 statfs 系统调用来获取文件系统信息。

【注意】确保传入的参数正确

【举例】无

十七、POWOFF

1) 概述

POWOFF 模块提供了一套统一的关机回调管理机制，主要包括回调注册、执行以及资源清理三个功能。用户可以通过注册各类关机前需执行的操作（如保存数据、释放资源等），在系统关机时统一调用这些回调函数，进而完成必要的关机准备工作。整个模块可以确保在关机流程中，各个模块都能有序、安全地完成清理和关闭操作。

2) API 参考

17.2.1 qm_register_shutdown_callback

【描述】该函数用于注册关机回调函数，即将用户提供的回调函数加入到关机回调链表中。注册的回调函数将在系统关机时依次被调用，用于完成资源释放、数据保存等关机前的必要操作。

【语法】int qm_register_shutdown_callback(int (callback)(void))

【参数】

参数名称	描述	输入/输出
Callback	函数指针，指向一个返回 int 类型的回调函数（输入）。回调函数返回 0 表示成功，非 0 表示失败	输入

【返回值】

返回值	描述
0	成功注册回调函数
-1	注册失败

【需求】使用前必须确保传入的回调函数有效，且回调函数中应正确处理关机前的操作。

【注意】如果传入的回调函数为 NULL，函数会输出错误提示并返回-1；新注册的回调函数会被插入到链表头部，执行顺序为逆注册顺序。

【举例】无

17.2.2 qm_execute_shutdown_callbacks

【描述】该函数用于遍历并执行所有已注册的关机回调函数。调用过程中，会顺序调用链表中各回调函数，以确保在关机时完成所有必要的清理或保存工作。如果任何一个回调执行失败，函数会输出错误信息，并最终返回失败标志。

【语法】int qm_execute_shutdown_callbacks(void)

【参数】无

【返回值】

返回值	描述
0	所有回调函数均成功执行
-1	至少有一个回调函数执行失败

【需求】在系统关机前调用此函数，确保所有关机前的操作得以依次执行。

【注意】即使某个回调执行失败，也不会阻止后续回调的执行；确保各个回调函数内部能够正确处理错误，避免影响整体流程。

【举例】无

17.2.3 qm_shutdown_callbacks_cleanup

【描述】该函数用于清理关机回调链表所占用的内存资源。它遍历整个回调链表并释放每个节点，最后将全局链表指针置空，从而防止内存泄露。

【语法】void qm_shutdown_callbacks_cleanup(void)

【参数】无

【返回值】无

【需求】在关机回调执行完成或不再需要时调用此函数，以释放所有注册的回调函数节点内存。

【注意】调用后，所有之前注册的回调函数信息将被清空，后续如果需要再次使用，则必须重新注册。

【举例】无

十八、IIO

1) 概述

提供 IIO 子系统在用户空间的标准 API 接口。通过遍历 "/sys/bus/iio/devices/" 目录下的各个设备文件夹，根据给定的设备类型前缀过滤、读取设备中的名称，然后与用户提供的设备名称进行匹配，若匹配成功，则返回设备编号。

2) API 参考

18.2.1 qm_iio_find_type_by_name

【描述】用于扫描 IIO 设备目录，查找设备名称与传入参数匹配的设备。

【语法】int qm_iio_find_type_by_name(const char device_name, const char type)

【参数】

参数名称	描述	输入/输出
const char device_name	待查找的 IIO 设备名称	输入
const char type	设备类型	输入

【返回值】

返回值	描述
整数	设备编号
负数	失败

【需求】 "/sys/bus/iio/devices/" 目录下应有对应设备文件夹；程序需要有足够权限访问这些 sysfs 目录和文件；系统中必须存在 IIO 设备。

【注意】传入的 device_name 必须与设备的实际名称严格匹配，否则即使设备存在也可能返回错误；在读取设备 “name” 文件过程中，若路径因 snprintf 被截断或文件读取失败，函数将输出错误提示信息并继续查找其他设备。

【举例】qm_iio_find_type_by_name("adc", IIO_TYPE_DEVICE)

十九、SYSFS

1) 概述

提供文件系统在用户空间的操作接口，用于对内核设备参数进行读写。

2) API 参考

19.2.1 qm_write_sysfs_int

【描述】将一个整数值写入到指定的 sysfs 文件中。

【语法】int qm_write_sysfs_int(const char *filename, const char *basedir, int val)

【参数】

参数名称	描述	输入/输出
const char *filename	文件名	输入
const char *basedir	目录路径	输入
int val	整数值	输入

【返回值】

返回值	描述
自然数	成功
负数	失败

【需求】 要求进程对所写入目录具有写权限；目标 sysfs 文件须支持整数写入格式。

【注意】写入成功后，本函数不验证写入数据是否正确写入。

【举例】qm_write_sysfs_int("brightness", "/sys/class/backlight/intel_backlight", 200)

19.2.2 qm_write_sysfs_int_and_verify

【描述】将整数值写入指定的 sysfs 文件，并通过读取回写回的数据进行验证，确保写入结果正确。

【语法】int qm_write_sysfs_int_and_verify(const char *filename, const char *basedir, int val)

【参数】

参数名称	描述	输入/输出
const char *filename	文件名	输入
const char *basedir	目录路径	输入
int val	整数值	输入

【返回值】

返回值	描述

自然数	成功
负数	失败

【需求】 目标 sysfs 文件必须能正确读写整数格式的数据，并且读操作不受权限或平台限制。

【注意】由于增加了验证步骤，可能比单纯写入操作耗时略长；切勿在对性能要求极高的场合滥用验证操作。

【举例】qm_write_sysfs_int_and_verify("brightness", "/sys/class/backlight/intel_backlight", 200);

19.2.3 qm_write_sysfs_string_and_verify

【描述】将一个字符串写入指定的 sysfs 文件，并通过重新读取文件验证写入的字符串是否正确。

【语法】int qm_write_sysfs_string_and_verify(const char *filename, const char *basedir, const char *val)

【参数】

参数名称	描述	输入/输出
const char *filename	文件名	输入
const char *basedir	目录路径	输入
const char *val	字符串	输入

【返回值】

返回值	描述
自然数	成功
负数	失败

【需求】目标 sysfs 文件须支持字符串操作，并能正确返回写入数据。

【注意】无

【举例】qm_write_sysfs_string_and_verify("mode", "/sys/devices/platform/device", "auto")

19.2.4 qm_write_sysfs_string

【描述】将一个字符串写入到指定的 sysfs 文件中。

【语法】int qm_write_sysfs_string(const char *filename, const char *basedir, const char *val)

【参数】

参数名称	描述	输入/输出
const char *filename	文件名	输入
const char *basedir	目录路径	输入
const char *val	字符串	输入

【返回值】

返回值	描述
自然数	成功
负数	失败

【需求】写入操作要求目标文件具有写入权限，且格式为字符串。

【注意】无

【举例】qm_write_sysfs_string("name", "/sys/devices/platform/device", "sensor");

19.2.5 qm_read_sysfs_posint

【描述】从指定的 sysfs 文件中读取一个正整数。

【语法】int qm_read_sysfs_posint(const char *filename, const char *basedir)

【参数】

参数名称	描述	输入/输出
const char *filename	文件名	输入
const char *basedir	目录路径	输入

【返回值】

返回值	描述
正整数	成功
负数	失败

【需求】目标文件应包含以文本形式存储的整数数据，并且格式规范。

【注意】读取过程中可能因格式问题或权限问题导致解析错误。

【举例】无

19.2.6 qm_read_sysfs_float

【描述】从指定的 sysfs 文件中读取一个浮点数，并将读取结果存放在调用者提供的变量中。

【语法】int qm_read_sysfs_float(const char *filename, const char *basedir, float *val)

【参数】

参数名称	描述	输入/输出
const char *filename	文件名	输入
const char *basedir	目标路径	输入
float *val	用于存储读取的浮点数	输出

【返回值】

返回值	描述
自然数	成功
负数	失败

【需求】要求目标 sysfs 文件必须能正确显示浮点数格式（例如温度、频率等数据）。

【注意】调用前需确保 val 指针非空，否则可能引发运行时错误；读取过程中注意文件中存在多余空白或换行符可能影响解析。

【举例】qm_read_sysfs_float("temp", "/sys/class/thermal", &temp)

19.2.7 qm_read_sysfs_string

【描述】从指定的 sysfs 文件中读取一个字符串，并把读到的内容存入调用者提供的缓冲区中。

【语法】int qm_read_sysfs_string(const char *filename, const char *basedir, char *str)

【参数】

参数名称	描述	输入/输出
const char *filename	文件名	输入
const char *basedir	目录路径	输入
const char *str	用于存储读取的字符串	输出

【返回值】

返回值	描述
自然数	成功
负数	失败

【需求】目标文件内的数据应为文本字符串形式。

【注意】调用者必须确保 str 缓冲区大小足够，以免造成截断或内存问题。

【举例】无

二十、HIERARCHICAL FILTER

1) 概述

主要实现分层过滤器的数据结构和 API 接口，用于实现数据平滑、趋势跟踪和异常值检测的综合处理。

2) API 参考

20.2.1 qm_hierarchical_filter_init

【描述】初始化分层过滤器。该函数根据传入的配置参数，初始化过滤器中包含的 IIR 滤波器、指数平滑和异常值检测模块，并为各模块分配所需内存。

【语法】int qm_hierarchical_filter_init(struct hierarchical_filter *filter,const struct hierarchical_filter_config *config)

【参数】

参数名称	描述	输入/输出
struct hierarchical_filter *filter	指向待初始化的分层过滤器	输入
const struct hierarchical_filter_config *config	分层过滤器的配置结构体	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】调用前确保传入的 filter 和 config 非空，并确保配置参数正确。

【注意】初始化失败需要检查并处理可能的内存泄露；确保 config 中各参数合法。

【举例】无

20.2.2 qm_hierarchical_filter_process_float

【描述】处理浮点型输入数据，并返回经过分层滤波器处理后的滤波值。

【语法】float qm_hierarchical_filter_process_float(struct hierarchical_filter *filter,float new_sample)

【参数】

参数名称	描述	输入/输出
struct hierarchical_filter *filter	指向已初始化的分层滤波器	输入
float new_sample	输入样本	输入

【返回值】

返回值	描述
经过处理后的浮点数值	成功
输入样本	失败

【需求】调用前需保证滤波器已经正确初始化。

【注意】内部首先调用异常值检测，若输入样本为异常，则用均值替代。

【举例】无

20.2.3 qm_hierarchical_filter_get_status

【描述】获取当前分层过滤器的状态信息，包括当前水平值、趋势、初始化状态及最新异常值检测的次数。

【语法】int qm_hierarchical_filter_get_status(struct hierarchical_filter *filter,struct hierarchical_filter_status *status)

【参数】

参数名称	描述	输入/输出
struct hierarchical_filter *filter	指向已初始化的分层滤波器	输入
struct hierarchical_filter_status *status	指向分层过滤器，填充当前状态信息	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】过滤器必须已经初始化且在使用过程中产生有效状态信息。

【注意】该函数内部会计算异常值的树林，使用缓冲区数据求均值和标准差；传参须合法。

【举例】无

20.2.4 qm_hierarchical_filter_destory

【描述】销毁分层过滤器，释放在初始化和使用过程中分配的内存，并清空数据。

【语法】void qm_hierarchical_filter_destory(struct hierarchical_filter *filter)

【参数】

参数名称	描述	输入/输出
struct hierarchical_filter *filte	指向带销毁的分层过滤器	输入

【返回值】无

【需求】在不需要过滤器时调用，避免内存泄露。

【注意】在销毁后，filter 结构体中的内容将被清空；调用后不要再次使用该过滤器。

【举例】无

二十一、BATTERY

1) 概述

提供完整的电池管理解决方案，实现了电池状态的实时检测和处理，允许上层应用通过回调机制及时响应电池事件，从而有效管理电池的运行状态。

2) API 参考

21.2.1 qm_bat_register_event_callback

【描述】注册电池事件回调函数。该函数允许用户为不同类型的电池事件（如电量变化、充电状态变化、低电量或电量危急警告）注册回调函数。当相应的电池事件发生时，系统会调用这些回调函数并传递事件类型、事件值和用户自定义数据。

【语法】int qm_bat_register_event_callback(bat_event_type_t type, bat_event_callback_t callback, void user_data)

【参数】

参数名称	描述	输入/输出
bat_event_type_t type	事件类型	输入
bat_event_callback_t callback	事件回调函数	输入
void user_data	用户数据指针	输入

【返回值】

返回值	描述

0	成功
负数	失败

【需求】调用前应确保事件类型合法并且回调函数非空。

【注意】同一事件类型中同一回调函数不能被重复注册；内部使用互斥锁保护回调数组，确保线程安全。

【举例】无

21.2.2 qm_bat_unregister_event_callback

【描述】注销之前注册的电池事件回调函数。调用此函数后，指定事件类型下对应的回调函数将不再收到电池状态的通知。

【语法】int qm_bat_unregister_event_callback(bat_event_type_t type, bat_event_callback_t callback, void user_data)

【参数】

参数名称	描述	输入/输出
bat_event_type_t type	事件类型	输入
bat_event_callback_t callback	事件回调函数	输入
void user_data	用户数据指针	输入

【返回值】

返回值	描述
0	成功
负数	失败

【需求】要注销的回调必须之前已经成功注册，并且在调用时仍处于有效状态。

【注意】注销过程中会同步等待可能正在执行的回调函数结束；系统内部会更新回调数组状态，并在最后一个回调注销时停止电池状态监控。

【举例】无

21.2.3 qm_bat_get_charging_status

【描述】获取当前电池的充电状态。该函数通过读取系统电源供电状态来判断电池是否处于充电状态。

【语法】int qm_bat_get_charging_status(void)

【参数】无

【返回值】

返回值	描述
1	正在充电
0	未充电

【需求】系统必须正确设置供电状态文件（如 /sys/class/power_supply/battery/status），且进程具有读取该文件的权限。

【注意】若读取状态失败，函数默认返回未充电状态；返回的充电状态为整型值，用户可据此进行状态判断。

【举例】无

21.2.4 qm_bat_is_full_capacity

【描述】判断电池是否已充满。函数检查电池当前容量是否达到 100%。

【语法】bool qm_bat_is_full_capacity(void)

【参数】无

【返回值】

返回值	描述
True	已充满
False	未充满

【需求】系统中必须能正常读取电池容量信息，否则判断可能不准确。

【注意】该函数仅依据读取到的电池容量数值进行判断。

【举例】无

21.2.5 qm_bat_get_current_capacity

【描述】获取当前电池容量（百分比）。该函数返回处理后的电池容量值，范围为 0～100。

【语法】int qm_bat_get_current_capacity(void)

【参数】无

【返回值】

返回值	描述
0-100	成功，电池容量
-1	失败

【需求】系统需能通过对应 sysfs 文件读取到正确的电池原始数据，并由内部机制对数据进行滤波处理以提高稳定性。

【注意】内部可能调用分层滤波器对原始数据进行处理后返回。

【举例】无

二十二、UEVENT

1) 概述

提供 uevent 的用户空间接口，支持用户通过注册回调函数获取设备状态变化、添加或移除等事件信息。

2) API 参考

22.2.1 qm_uevent_register_filter

【描述】注册设备过滤器。该函数根据传入的过滤器配置（包含子系统、设备类型、设备名称和设备路径的匹配模式）建立一个过滤器，用于筛选感兴趣的 uevent 事件。注册成功后，过滤器将参与后续 uevent 事件的匹配。

【语法】int qm_uevent_register_filter(const qm_uevent_filter_t *filter)

【参数】

参数名称	描述	输入/输出
const qm_uevent_filter_t *filter	指向 qm_uevent_filter_t 结构体的指针	输入

【返回值】

返回值	描述
非负整数	成功
-1	失败

【需求】调用前确保传入的 filter 指针非空，并且过滤器的各字段符合预期的格式。

【注意】多个过滤器可同时注册，但系统支持的最大过滤器数有限（MAX_FILTERS）；若过滤器资源耗尽，会返回-ENOBUFS。

【举例】无

22.2.2 qm_uevent_unregister_filter

【描述】注销已注册的设备过滤器。该函数通过提供的过滤器 ID 停用并删除相应的过滤器，不再进行 uevent 事件的匹配。

【语法】int qm_uevent_unregister_filter(int filter_id)

【参数】

参数名称	描述	输入/输出
int filter_id	需要注销的过滤器 ID	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】确保在注销之前，该过滤器已成功注册，并且过滤器 ID 正确。

【注意】注销过滤器后，相关的过滤条件将不再影响 uevent 事件回调；多线程环境下，注销操作受到内部互斥锁保护。

【举例】无

22.2.3 qm_uevent_register_callback

【描述】注册 uevent 事件回调函数。该函数允许用户注册一个回调函数，当系统接收到符合条件的 uevent 事件时，该回调函数将被调用，并传递事件详细信息和用户自定义数据。

【语法】int qm_uevent_register_callback(qm_uevent_callback_t callback, void *user_data);

【参数】

参数名称	描述	输入/输出
qm_uevent_callback_t callback	事件回调函数指针	输入
void *user_data	用户数据指针	输入

【返回值】

返回值	描述
非负整数	成功
-1	失败

【需求】确保传入的回调函数非空，并且系统已经初始化 uevent 监控上下文。

【注意】系统支持的回调函数数量有限（MAX_CALLBACKS），超过限制将返回 -ENOBUFS；注册过程中内部使用互斥锁保证线程安全。

【举例】无

22.2.4 qm_uevent_unregister_callback

【描述】注销 uevent 事件回调函数。通过提供的回调 ID，该函数将取消对应回调的注册，使其不再被调用。

【语法】int qm_uevent_unregister_callback(int callback_id)

【参数】

参数名称	描述	输入/输出
int callback_id	需要注销的回调 ID	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】确保在注销之前，回调已经成功注册。

【注意】注销操作过程中将更新内部回调数组，并使用互斥锁保证线程安全；注销后，该回调的相关资源将被清空。

【举例】无

22.2.5 qm_uevent_get_filter_count

【描述】获取当前已注册的设备过滤器数量。

【语法】int qm_uevent_get_filter_count(void)

【参数】无

【返回值】

返回值	描述
0	失败
正整数	成功，过滤器数量

【需求】确保 uevent 监控系统已初始化。

【注意】内部操作受互斥锁保护。

【举例】无

22.2.6 qm_uevent_get_callback_count

【描述】获取当前已注册的 uevent 回调函数数量。

【语法】int qm_uevent_get_callback_count(void)

【参数】无

【返回值】

返回值	描述
0	失败
正整数	成功

【需求】确保 uevent 监控系统已初始化。

【注意】内部操作受互斥锁保护。

【举例】无