目录

使用二次封装库

1.获取二次封装库：http://wiki.armbbs.net/tqwiki/public/docs/

2.解压二次封装库

将下载到的二次封装库压缩包，解压到 ubuntu 系统中，执行命令为:

```
#tar xvjf linux_lib_VX.x.tar.bz2 -C /
```

3.QT 使用 sdk 库：

在.pro 文件最后添加：

```
LIBS += -L/opt/EmbedSky/linux_lib/lib -ltq++
INCLUDEPATH += /opt/EmbedSky/linux_lib/include
DEPENDPATH += /opt/EmbedSky/linux_lib/include
PRE_TARGETDEPS += /opt/EmbedSky/linux_lib/lib/libtq++.a
```

4.交叉编译使用 sdk 库：

使用 gcc 编译 test.c 文件:

```
arm-linux-gnueabihf-gcc -I/opt/EmbedSky/linux_lib/include
-L/opt/EmbedSky/linux_lib/lib test.c -o test -ltq
```

使用 g++编译 test.cpp 文件:

```
arm-linux-gnueabihf-g++ -I/opt/EmbedSky/linux_lib/include
-L/opt/EmbedSky/linux_lib/lib test.cpp -o test -ltq++
```

常用接口

头文件：

tq_common.h

函数接口：

函数	功能
int check_file(char* filename);	检查文件是否存在
int write_file_str(char* filename, char* value, int size);	往文件里写字符串
int read_file_str(char* filename, char* value, int size);	读取文件内容(字符串)
int write_file_int(char* filename, unsigned int value);	往文件写入整型
int read_file_int(char* filename, int* value);	读取文件内容(整型)
int getPidByName(char* task_name, int* pid);	通过进程名字获取进程号
int getNameByPid(int pid, char *task_name);	通过进程号获取进程名字
const char* get_libversion(void);	获取库版本号
char* get_boardinfo(void);	获取板卡信息

check_file

函数： int check_file(char* filename);

描述：

检查文件是否存在

参数：

参数名	类型	说明
filename	char *	文件路径

示例：

```
int ret = check_file("/test");
printf("ret:%d\n",ret);
```

返回参数说明：

返回值	类型	说明
1	int	存在
0	int	不存在/打开失败

write_file_str

函数：

```
int write_file_str(char  filename, char  value, int size);
```

描述：

往文件里写字符串

参数：

参数名	类型	说明
filename	char *	文件路径
value	char *	写入的内容
size	int	内容大小

示例：

```
char buff[] = "out";
int ret = write_file_str("/test",buff,sizeof(buff));
printf("ret :%d\n",ret);
```

返回参数说明：

返回值	类型	说明
>0	int	成功返回写入的大小
-1	int	打开失败
-2	int	写入失败

read_file_str

函数：

```
int read_file_str(char  filename, char  value, int size);
```

描述：

读取文件内容

参数：

参数名	类型	说明
filename	char *	文件路径
value	char *	用于存放读取到的文件内容
size	int	内容大小

示例：

```
char buff[] = "out";
int ret = write_file_str("/test",buff,sizeof(buff));
printf("ret :%d\n",ret);
```

返回参数说明：

返回值	类型	说明
>0	int	读取到的大小
-1	int	打开失败
-2		

write_file_int

函数：

```
int write_file_int(char* filename, unsigned int value);
```

描述：

往文件写入整型

参数：

参数名	类型	说明
filename	char *	文件路径
value	unsigned int	存放要写入文件的内容

示例：

```
int ret  = write_file_int("/sys/class/leds/led4/brightress",1);
printf("ret:%d\n",ret);
```

返回参数说明：

返回值	类型	说明
>0	int	写入的大小
-1	int	打开失败
-2	int	写入失败

read_file_int

函数：

```
int read_file_int(char  filename, int  value);
```

描述：

检查文件是否存在

参数：

参数名	类型	说明
filename	char *	文件路径
value	unsigned int	存放读取到文件的内容

示例：

```
int buff;
int ret = read_file_int("/test",&buff);
printf("ret:%d,buff:%d\n",ret,buff);
```

返回参数说明：

返回值	类型	说明
>0	int	读取到的大小
-1	int	打开失败
-2	int	读取失败

getPidByName

函数：

```
int getPidByName(char  task_name, int  pid);
```

描述：

通过进程名字获取进程号

参数：

参数名	类型	说明
task_name	char *	进程名字
pid	int *	用于存放进程号

示例：

```
int pid;
int ret = getPidByName("test",&pid);
printf("ret = %d\n",ret);
```

返回参数说明：

返回值	类型	说明
0	int	成功
-1	int	失败

getNameByPid

函数：

```
int getNameByPid(int pid, char *task_name);
```

描述：

通过进程号获取进程名字

参数：

参数名	类型	说明
pid	int *	进程号
task_name	char *	用于存放进程名字

示例：

```
char buff[100];
memset(buff,0x0,sizeof(buff));
int ret = getNameByPid(962,buff);
printf("ret = %d,buff = %s\n",ret,buff);
```

返回参数说明：

返回值	类型	说明
0	int	成功
-1	int	失败

get_libversion

函数：

```
const char* get_libversion(void);
```

描述：

获取二次封装库版本

参数：

无

示例：

```
printf("tq linux lib version %s\n",get_libversion());
```

返回参数说明：

返回库版本字符串

get_boardinfo

函数：

```
char* get_boardinfo(void);
```

描述：

获取板卡信息

参数：

无

示例：

```
char* board_info =get_boardinfo() ;
if(board_info == NULL) {
    printf(“board info no found\n”);
} else {
    printf(“board info is %s\n”,board_info);
}
```

返回参数说明：

返回值	类型	说明
!NULL	char*	成功，返回板卡信息
NULL	char*	未获取到板卡信息

GPIO 操作

头文件：

tq_gpio.h

函数接口：

函数	功能
int tq_set_gpio(int pin,int val);	设置 GPIO 高低电平
int tq_get_gpio(int pin);	获得 GPIO 外部输入的电平状态(非阻塞)
int tq_get_gpio_poll(int pin, int edge, int timeout_ms);	获得 GPIO 外部输入的电平状态(中断阻塞)
int gpio_to_pin(unsigned char gpiochip, unsigned char gpioid);	获取 gpio pin 号

tq_set_gpio

函数：

```
int tq_set_gpio(int pin,int val);
```

描述：

设置 GPIO 高低电平

参数：

参数名	类型	说明
pin	int	引脚编号
val	int	电平状态　1：高电平　0：低电平

示例：

```
int temp = tq_set_gpio(80,1);
if(temp<0){
    printf("tq_set_gpio fail\n");
}else {
    printf("tq_set_gpio success\n");
}
```

返回参数说明：

返回值	类型	说明
0	int	成功
-1	int	失败

tq_get_gpio

函数：

```
int tq_get_gpio(int pin);
```

描述：

获得 GPIO 外部输入的电平状态

参数：

参数名	类型	说明
pin	int	引脚编号

示例：

```
int ret;
ret = tq_get_gpio(80);
if(ret<0){
   printf("tq_get_gpio failed:%d\n",ret);
  }else{
       printf("tq_get_gpio success:%d\n",ret);
  }
  return;
```

返回参数说明：

返回值	类型	说明
0	int	低电平
1	int	高电平
-1	int	失败

tq_get_gpio_poll

函数：

```
int tq_get_gpio_poll(int pin, int edge, int timeout_ms);
```

描述：

获得 GPIO 外部输入的电平状态(中断阻塞)

参数：

参数名	类型	说明
pin	int	引脚编号
edge	int	触发类型，可选值为: 0:不阻塞。 1:上升沿触发。 2:下降沿触发。 3:上升/下降沿触发
timeout_m	int	阻塞时间，单位 ms，-1 为一直阻塞，0 为不阻塞

示例：

```
int ret;
ret = tq_get_gpio_poll(80,1,200);
if(ret<0){
   printf("tq_get_gpio failed:%d\n",ret);
  }else{
       printf("tq_get_gpio success:%d\n",ret);
  }
  return;
```

返回参数说明：

返回值	类型	说明
0	int	低电平
1	int	高电平
-1	int	失败

gpio_to_pin

函数：

int gpio_to_pin(unsigned char gpiochip, unsigned char gpioid)

描述：

将 gpio 网络号转换为可操作的 gpio id;

参数：

参数名	类型	说明
gpiochip	unsigned char	gpio 组号，注:组号以 0 开始，如果搜索到原理图中无 gpio0 的组，那么控制 gpio1 时需填入 0
gpioid	unsigned char	gpio 组内的 id 号

示例：

```
int pin = gpio_to_pin(1,2); //AM335X:获取 gpio1_2 的 id 号,TQIMX6Q:获取 gpio2_2 的 id 号(因为 IMX6Q 的 gpio 组是以 1 开始)
int result = tq_get_gpio_poll(pin,3,-1);
tq_set_gpio(pin,1); //输出高电平
```

返回参数说明：

返回值	类型	说明
>=0	int	转换后的 id 号
-1	int	传递参数错误
-2	int	没有搜索到相应的 gpio 组

UART 操作

头文件：

tq_uart.h

函数接口：

函数	功能
int uart_init(char* dev, int nBaud, int nBits, char nEvent, int nStop);	初始化 uart 参数
int uart_write_data(char *dev_name,char *buff,int len);	发送串口数据
int uart_start_receive(char *dev_name);	启动接收数据
int uart_read_data(int fd, char *buff,int len, int timeout_ms);	读取接收到的数据
void uart_stop_receive(int fd);	停止接收数据

uart_init

函数：

```
int uart_init(char* dev, int nBaud, int nBits, char nEvent, int nStop);
```

描述：

初始化串口

参数：

参数名	类型	说明
dev	char *	串口名称，如”/dev/ttySAC3”
nBaud	int	波特率，可取值:300 600 1200 2400 4800 9600 19200 38400 57600 115200…
nBits	int	数据位数 7 或者 8
nEvent	char	‘O’ 奇校验； ‘E’ 偶校验； ‘N’ 无校验
nStop	int	停止位数 1 或者 2

示例：

```
int fd;
char *dev_name = "/dev/ttySAC3";
if(fd = uart_init(dev_name,115200,8, 'E',1)<0)
    printf("set_opt error\n");
else
    printf("uart_init success\n");
```

返回参数说明：

返回值	类型	说明
0	int	初始化成功
-1	int	初始化失败

uart_write_data

函数：

```
int uart_write_data(char  dev_name,char  buff,int len);
```

描述：

往串口写数据

参数：

参数名	类型	说明
dev_name	char *	串口名称， 如”/dev/ttySAC3”
buff	char *	存放要写入的内容
len	int	内容大小

示例：

```
char *dev_name = "/dev/ttySAC3";
char buff[64] = "123456789" ;
int ret = uart_write_data(dev_name,buff,sizeof(buff));
      //发送数据到 dev_name
if(ret < 0)
   printf("send err\n");
else
   printf("write_data success\n");
```

返回参数说明：

返回值	类型	说明
>0	int	写入的大小
-1	int	写入失败
-5	int	打开失败

uart_start_receive

函数：

```
int uart_start_receive(char *dev_name);
```

描述：

往串口写数据

参数：

参数名	类型	说明
dev_name	char *	串口名称,如”/dev/ttySAC3”

示例：

```
char *dev_name = "/dev/ttySAC3";
read_fd = uart_start_receive(dev_name);
if(read_fd < 0)
    printf("uart_start_receive err\n");
else
    printf("start_receive success\n");
```

返回参数说明：

返回值	类型	说明
>0	int	写入的大小
-5	int	打开失败

uart_read_data

函数：

```
int uart_read_data(int fd, char *buff,int len, int timeout_ms);
```

描述：

读取串口信息

参数：

参数名	类型	说明
fd	char *	串口名称， 如”/dev/ttySAC3”
buff	char *	存放要写入的内容
len	int	内容大小
timeout_ms	int	设置超时时间,/ms

示例：

```
char str[64];
memset(str,0,sizeof(str));
int ret = uart_read_data(read_fd,str,sizeof(str), 100);     //从串口接收数据
if(ret < 0)
   printf("receive err\n");
else
   printf("receive success:%s\n",str);
```

返回参数说明：

返回值	类型	说明
>0	int	返回读入的大小
-1	int	读取失败
-2	int	读取超时

uart_stop_receive

函数：

```
void uart_stop_receive(int fd);
```

描述：

停止接收信息

参数：

参数名	类型	说明
fd	int	文件句柄

示例：

```
uart_stop_receive(read_fd);
```

返回参数说明：

返回值	类型	说明
无	无	无

网络操作

头文件：

tq_network.h

函数接口：

函数	功能
int get_ip(char* dev, char* ip,int size);	获取指定网卡设备的 ip 地址
int get_netmask(char* dev, char* mask,int size);	获取指定网卡设备的子网掩码
int get_gateway(char* dev, char* gway,int size);	获取指定网卡设备的网关
int get_mac(char* dev, char* mac,int size);	获取指定网卡设备的 mac 地址
int check_ping(const char *target,unsigned int timeout_ms);	网络检测函数，类似于 ping 命令(不能选择网卡设备)
int check_ping2(const char *iface, const char *target, unsigned int timeout_ms);	网络检测函数，类似于 ping 命令(可选择网卡设备)

get_ip

函数：

```
int get_ip(char  dev, char  ip,int size);
```

描述：

获取指定网卡设备的 ip 地址

参数：

参数名	类型	说明
dev	const char *	网卡设备名,如“eth0”,”ens33”
ip	char *	获取后的 ip 填充位置
size	int	传入 ip 的大小

示例：

```
char ip[IP_SIZE];
char *test_eth = "eth0";
int len=0;
len = get_ip(test_eth, ip, sizeof(ip));
if(len < 0){
    printf("get ip error %d\n",len);
else
    printf("local %s ip: %s len: %d \n", test_eth, ip,len);
```

返回参数说明：

返回值	类型	说明
>0	int	成功,填充 ip 的长度
0	int	没有找到设备节点
-1	int	失败，无该设备节点
-2	int	失败,形参 ip 的空间长度不够,需要的最小长度见返回 ip 的首字节

get_netmask

函数：

```
int get_netmask(char  dev, char  mask,int size);
```

描述：

获取指定网卡设备的子网掩码

参数：

参数名	类型	说明
dev	char *	网卡设备名,如“eth0”,”enss33”
mask	char *	获取后的子网掩码填充位置
size	int	传入 mask 的大小

示例：

```
char ip[IP_SIZE];
char *test_eth = "eth0";
int len=0;
len = get_netmask(test_eth, ip, sizeof(ip));
if(len < 0)
    printf("get netmask error: %d\n", len);
else
    printf("local %s mask: %s len: %d \n", test_eth, ip,len);
```

返回参数说明：

返回值	类型	说明
>0	int	成功,填充 mask 的长度
0	int	没有找到设备节点
-1	int	失败，无该设备节点
-2	int	失败,形参 mask 的空间长度不够,需要的最小长度见返回 mask 的首字节

get_gateway

函数：

```
int get_gateway(char  dev, char  gway,int size);
```

描述：

获取指定网卡设备的网关

参数：

参数名	类型	说明
dev	char *	网卡设备名,如“eth0”,”enss33”
gway	char *	获取后的网关填充位置
size	int	传入 mask 的大小

示例：

```
char ip[IP_SIZE];
char *test_eth = "eth0";
int len=0;
len = get_gateway(test_eth, ip, sizeof(ip));
if(len < 0)
    printf("get gateway error:%d\n",len);
else
    printf("local %s gateway: %s len: %d \n", test_eth, ip,len);
```

返回参数说明：

返回值	类型	说明
>0	int	成功,填充 gway 的长度
0	int	没有找到设备节点
-1	int	失败，无该设备节点
-2	int	失败,形参 gway 的空间长度不够,需要的最小长度见返回 gway 的首字节

get_mac

函数： int get_mac(char dev, char mac,int size);

描述：

获取指定网卡设备的 mac 地址

参数：

参数名	类型	说明
dev	char *	网卡设备名,如“eth0”,”enss33”
mac	char *	获取后的 mac 填充位置
size	int	传入 mac 的大小

示例：

```
char mac[MAC_SIZE];
char *test_eth = "eth0";
int len=0;
len = get_mac(test_eth, mac, sizeof(mac));
if(len < 0)
    printf("get mac error:%d\n",len);
else
    printf("local %s mac: %s len: %d \n", test_eth, mac,len);
```

返回参数说明：

返回值	类型	说明
>0	int	成功,填充 mac 的长度
0	int	没有找到设备节点
-1	int	失败，无该设备节点
-2	int	失败,形参 mac 的空间长度不够,需要的最小长度见返回 mac 的首字节

check_ping

函数：

```
int check_ping(const char *target,unsigned int timeout_ms);
```

描述：

网络检测函数，类似于 ping 命令

参数：

参数名	类型	说明
target	const char *	目标地址域名或者 ip
timeout_ms	unsigned int	检测超时，毫秒

示例：

```
const char *test_domain = "www.qq.com";
char *test_eth = "eth0";
int ret;
ret = check_ping(test_domain,20);
    if(ret < 1){
       printf("the ping is error %d\n",ret);
    }else
       printf("the ping is ok!\n");
```

返回参数说明：

返回值	类型	说明
>0	int	成功
0	int	超时
-1	int	失败

check_ping2

函数：

int check_ping2 (const char iface, const char target, unsigned int timeout_ms);

描述：

网络检测函数，类似于 ping 命令（指定端口）

参数：

参数名	类型	说明
iface	const char *	网口名称，如“eth0”
target	const char *	目标地址域名或者 ip
timeout_ms	unsigned int	检测超时，毫秒

示例：

```
const char *test_domain = "www.qq.com";
int ret;
ret = check_ping2("eth0",test_domain,20);
if(ret < 1)
    printf("the ping2 is error %d\n",ret);
else
    printf("the ping2 is ok!\n");
```

返回参数说明：

返回值	类型	说明
>0	int	成功
0	int	超时
-1	int	失败

CAN 操作

头文件：

tq_can.h

函数接口：

函数	功能
int check_can(char *dev);	检测系统中是否存在 can 设备
int init_can(const char * dev,int bitrate, int mode);	初始化 can 设备
int get_can_data(const char* dev, struct can_data* data,unsigned int timeout_ms);	用于提取 can 接收到的数据
int send_can_data(const char* dev, struct can_data data);	用于发送 can 数据
int close_can(const char * dev);	关闭 can 设备

init_can

函数：

```
int init_can(const char * dev,int bitrate, int mode);
```

描述：

初始化 can 设备

参数：

参数名	类型	说明
dev	char *	can 设备名如 can0/can1
bitrate	int	设置的波特率
mode	int	设置的模式(1/2/4/8)

模式说明：

值	说明
1	回环模式
2	监听模式
4	三重采样模式
8	单发模式

示例：

```
int ret;
ret = init_can("can0",125000,0);
if(ret < 0)
    printf("don't init the can0 device\n");
else
    printf("init the can0 device\n");
```

返回参数说明：

返回值	类型	说明
0	int	成功
-1	int	没有检测到设备
-2	int	初始化失败
-99	int	参数错误

get_can_data

函数：

int get_can_data(const char dev, struct can_data data,unsigned int timeout_ms);

描述：

用于提取 can 接收到的数据

参数：

参数名	类型	说明
dev	const char*	can 设备名如 can0/can1
data	struct can_data*	用于填充读取的数据
timeout_ms	unsigned int	用于设置阻塞时间

示例：

```
struct can_data revcan;
memset(&revcan,0x0,sizeof(revcan));
ret = get_can_data("can0",&revcan,500);
printf("get can0 data, please wait for the data to return....\n");
if(ret < 0)
    printf("get can0 data error!ret=%d\n",ret);
else {
    printf("\t rec : [%d] \t",revcan.dlc);
    for(i = 0;i < revcan.dlc; i++)
        printf("0x%02x ",revcan.data[i]);
    printf("\n");
}
```

返回参数说明：

返回值	类型	说明
>=0	int	成功，成功返回读取到的帧数
-1	int	打开失败
-2	int	读取超时
-98	int	未初始化
-99	int	参数有误

send_can_data

函数：

```
int send_can_data(const char* dev, struct can_data data);
```

描述：

用于发送 can 数据

参数：

参数名	类型	说明
dev	const char*	can 设备名如 can0/can1
data	struct can_data*	用于填充读取的数据

示例：

```
struct can_data sendcan = {
    .id = 0x01,
    .dlc = 8,
    .data={0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08},
};
int ret;
printf("sand can0 data.... \n");
ret = send_can_data("can0",sendcan);
if(ret < 0)
    printf("send can0 data error!\n");
else
    printf("send can0 data success!\n");
```

返回参数说明：

返回值	类型	说明
>0	int	成功，成功返回写入的帧数
-1	int	打开失败
-2	int	写入失败
-98	int	未初始化
-99	int	参数有误

close_can

函数：

```
int close_can(const char * dev);
```

描述：

关闭 can 设备

参数：

参数名	类型	说明
dev	const char*	can 设备名 如”can0”,”can1”

示例：

```
close_can("can0");
```

返回参数说明：

返回值	类型	说明
0	int	成功
-1	int	关闭失败
-98	int	未初始化
-99	int	参数有误

看门狗

头文件：

tq_wdg.h

函数接口：

函数	功能
int EnableWtd(int TimeOut_s);	开启看门狗
int FeedWtd(void);	喂狗
int DisableWtd(void);	关闭看门狗

EnableWtd

函数：

```
int EnableWtd(int TimeOut_s);
```

描述：

使能看门狗设备

参数：

参数名	类型	说明
TimeOut	int	指定看门狗的最大喂狗间隔（秒）

示例：

```
int ret = EnableWtd(5);
printf("ret:%d\n",ret);
```

返回参数说明：

返回值	类型	说明
0	int	成功
-1	int	打开失败/设备支持

FeedWtd

函数：

```
int FeedWtd(void);
```

描述：

使能看门狗设备

参数：

无

示例：

```
int ret = FeedWtd();
printf("ret:%d\n",ret);
```

返回参数说明：

返回值	类型	说明
0	int	成功
-1	int	打开失败/设备不支持

DisableWtd

函数：

```
int DisableWtd();
```

描述：

关闭看门狗

参数：

无

示例：

```
int ret = DisableWtd();
printf("ret:%d\n",ret);
```

返回参数说明：

返回值	类型	说明
0	int	成功
-1	int	打开失败/设备支持

UVC 摄像头

头文件：

tq_uvc.h

函数接口：

函数	功能
int uvc_check(char* dev);	检测摄像头是否存在
int uvc_init(char* dev_name, int width, int high, int fps, void** buff);	初始化摄像头
int uvc_args(int fd,int* width,int* high);	获取设备可支持的图像尺寸
int uvc_format(int fd,struct uvc_struct_format**);	获取设备可支持的格式
struct uvc_struct_Attributes uvc_Attributes(int fd);	查询设备属性
int uvc_capturing(int fd);	将已经捕获好视频的内存拉出已捕获视频的队列
int uvc_release(int fd,int index);	将空闲的内存加入可捕获视频的队列
int uvc_close(int fd,void** buff);	关闭摄像头

uvc_check

函数：

```
int uvc_check (char* dev);
```

描述：

检测摄像头是否存在

参数：

参数名	类型	说明
dev	char *	摄像头设备名称,例如:”/dev/video0”,”/dev/video2”

示例：

```
char *devname="/dev/video2";
int ret = uvc_check (devname);
if(ret < 0) {
    printf("uvc_check error\n");
}
```

返回参数说明：

返回值	类型	说明
0	int	摄像头存在
-1	int	摄像头不存在
-2	int	关闭失败
-99	int	传入参数有误

uvc_init

函数：

int uvc_init(char* dev_name, int width, int high, int fps, void ** buff);

描述：

初始化 usb 摄像头

参数：

参数名	类型	说明
dev_name	char *	摄像头设备名称,例如:”/dev/video0”,”/dev/video2”
width	int	宽
high	int	高
fps	int	帧率
buff	void**	缓存

示例：

```
int ret = 0;
void *buff;
char *devname="/dev/video2";
int width = 640;
int high = 480;
int fps = 30;
ret = uvc_init(devname,width,high,fps,&buff);
if(ret <0) {
    printf("init v4l2 faile:%d\n",ret);
}
```

返回参数说明：

返回值	类型	说明
>0	int	成功，返回设备文件描述符
-1	int	打开设备文件失败
-2	int	设置数据流参数失败
-3	int	设置视频的制式和帧格式

返回值	类型	说明
-4	int	内存申请失败
-5	int	查询申请到的内存失败
-6	int	将空闲的内存加入可捕获视频的队列失败
-7	int	打开视频流失败
-99	int	打开视频流失败

uvc_args

函数：

```
int uvc_args(int fd,int  width,int  high);
```

描述：

获取 USB 摄像头设备可支持的图像尺寸

参数：

参数名	类型	说明
fd	int	设备文件描述符(uvc_init 的返回值)
width	int*	用于存放设备当前分辨率的宽
high	int*	用于存放设备当前分辨率的高

示例：

```
int width_now=0;
int high_now=0;
int result=uvc_args(ret,&width_now,&high_now);
if(result >0){
    printf("width:%d\n",width_now);
    printf("high:%d\n",high_now);
}else {
    printf("get camera args failed\n");
}
```

返回参数说明：

返回值	类型	说明
0	int	获取成功
-1	int	获取失败
-98	int	未初始化
-99	int	传入参数有误

uvc_format

函数：

```
int uvc_format(int fd,struct uvc_format** format);
```

描述：

获取设备可支持的格式

参数：

参数名	类型	说明
fd	char *	设备文件描述符(uvc_init 的返回值)
format	uvc_format**	用于存放设备可支持的格式

示例：

```
struct uvc_struct_format *fromat=NULL;
int format_i=uvc_format(ret,& fromat);
if(format_i>0){
    for(int ii=0;ii<format_i;ii++){
        printf("format:%s\n", fromat ->format);
        fromat ++;
    }
}else
    printf("get format error\n");
```

返回参数说明：

返回值	类型	说明
>0	int	支持格式的数量
-98	int	未初始化
-99	int	参数有误

uvc_Attributes

函数：

struct uvc_Attributes uvc_Attributes(int fd);

描述：

查询设备属性

参数：

参数名	类型	说明
fd	int	设备文件描述符(uvc_init 的返回值)

示例：

```
struct uvc_struct_Attributes cap = uvc_Attributes (ret);
    if(cap.isError<0){
        printf("uvc_Attributes error\n");
        return;
    }
printf("Driver Name:%s\nCard Name:%s\nBus info:%s\nDriver Version:%s\n",cap.driverName,cap.cardName,cap.bus_info,cap.version);
```

返回参数说明：

返回结构体中的值	类型	说明
>0	int	成功
-1	int	查询失败
-98	int	未初始化
-99	int	参数有误

uvc_capturing

函数：

```
int uvc_capturing(int fd);
```

描述：

将已经捕获好视频的内存拉出已捕获视频的队列

参数：

参数名	类型	说明
fd	int	设备文件(uvc_init 的返回值)

示例：

```
int index=uvc_capturing(ret);
if(index<0)
    printf("uvc_capturing error\n");
```

返回参数说明：

返回值	类型	说明
>0	int	初始化成功
-1	int	内存拉出失败
-98	int	未初始化
-99	int	参数错误

uvc_release

函数：

```
int uvc_release(int fd,int index);
```

描述：

将空闲的内存加入可捕获视频的队列

参数：

参数名	类型	说明
fd	int	设备文件描述符(uvc_init 的返回值)
index	int	索引值(uvc_capturing 的返回值)

示例：

```
int result = uvc_release(ret,index);
if(result <0){
    printf("uvc_release error!\n");
}
```

返回参数说明：

返回值	类型	说明
0	int	成功
-1	int	加入失败
-98	int	未初始化
-99	int	参数有误

uvc_close

函数： int uvc_close(int fd,void** buff);

描述：

关闭 usb 摄像头

参数：

参数名	类型	说明
fd	int	设备文件描述符(uvc_init 的返回值)
buff	void**	缓存(与初始化函数 uvc_init 调用同一个)

示例：

```
int result = uvc_close(ret,&buff);
if(result<0)
    printf("uvc_close error\n");
```

返回参数说明：

返回值	类型	说明
0	int	成功
-1	int	关闭失败
-2	int	查询申请到的内存失败
-98	int	未初始化
-99	int	参数有误

存储(U 盘、SD 卡等)

头文件：

tq_disk.h

函数接口：

函数	功能
int get_udisk_count(void);	获取 u 盘个数
int get_udisk_path(int index, int part, char* buff, int size);	获取 u 盘挂载路径
int get_sdisk_path(int part, char* buff, int size);	获取 sd 卡挂载路径

get_udisk_count

函数：

```
int get_udisk_count(void);
```

描述：

获取 u 盘个数

参数：

参数名	类型	说明
无	无	无

示例：

```
int ucount = get_udisk_count();
printf("get_udisk_count:%d\n",ucount);
```

返回参数说明：

返回值	类型	说明
>=0	int	成功，返回 u 盘个数
-1	int	失败

get_udisk_path

函数：

```
int get_udisk_path(int index, int part, char* buff, int size);
```

描述：

获取 u 盘挂载路径

参数：

参数名	类型	说明
index	int	u 盘号
part	int	分区号
buff	char*	获取后挂载路径
size	int	buff 的空间大小

示例：

```
char buff[100]={0};
int ret;
int part =1;
ret = get_udisk_path(1,part,buff,sizeof(buff));
if(ret < 0){
    printf("error!!!\n");
}else
    printf("get_udisk_path is ok [%d]: %s\n",ret,buff);
```

返回参数说明：

返回值	类型	说明
>0	int	返回填充 buff 的长度
0	int	无挂载
-1	int	失败
-2	int	buff 大小错误

get_sdisk_path

函数：

```
int get_sdisk_path(int part, char* buff, int size);
```

描述：

获取 sd 卡挂载路径

参数：

参数名	类型	说明
part	int	分区号
buff	char*	获取后挂载路径
size	int	buff 的空间大小

示例：

```
char buff[100]={0};
int part=1;
int ret = get_sdisk_path(part,buff,sizeof(buff));
if(ret < 0){
    printf("error!!!\n");
}else
    printf("get_sdisk_path is ok [%d]: %s\n",ret,buff);
```

返回参数说明：

返回值	类型	说明
>0	int	返回填充 buff 的长度
0	int	无挂载
-1	int	失败

输入设备

头文件：

tq_getinput.h

函数接口：

函数	功能
int tq_getinput(char* touch_type,char* dev_name);	通过输入设备 name 获取设备节点

tq_getinput

函数：

```
int tq_getinput(char  touch_type,char  dev_name);
```

描述：

传入字符串判断触摸是哪个设备，将设备名写入指针地址

参数：

参数名	类型	说明
touch_type	char *	触摸类型，需要字符串，如：“Capacitance_ts”：电容触摸;“tq_hs0038”：红外遥控;“Resistance_ts”：电阻触摸；“gpio-keys”：按键
dev_name	char*	返回 touch_type 对应的设备节点

示例：

```
char dev_name[13];
int ret = tq_getinput("Capacitance_ts",dev_name);
if(ret<0){
    printf("tq_getinput error:%d\n",ret);
    return -1;
}
printf("Capacitance_ts dev is--->%s\n",dev_name);
```

返回参数说明：

返回值	类型	说明
0	int	成功
-1	int	打开临时文件失败
-2	int	读取临时文件失败
-99	int	参数有误

PWM 控制

头文件：

tq_pwm.h

函数接口：

函数	功能
int pwm_enable(int node, int enable);	输出 pwm 信号
int tq_set_pwm(int node,unsigned long period,unsigned long duty_cycle);	配置 pwm 输出参数

pwm_enable

函数：

```
int pwm_enable(int node, int enable);
```

描述：

PWM 开启/关闭

参数：

参数名	类型	说明
node	int	设备节点
enable	int	1：启动；0：关闭

示例：

```
int ret = pwm_enable(2,1);
printf("ret=%d\n",ret);
```

返回参数说明：

返回值	类型	说明
0	int	成功
-1	int	打开失败
-2	int	写入失败

tq_set_pwm

函数：

int tq_set_pwm(int node,unsigned long period,unsigned long duty_cycle);

描述：

设置 PWM 的周期、占空比

参数：

参数名	类型	说明
node	int	设备节点
period	unsigned long	pwm 周期(单位 ns)
duty_cycle	unsigned long	占空比(单位 ns)

示例：

```
int ret = tq_set_pwm(2,1000000,500000);
printf("ret = %d\n",ret);
```

返回参数说明：

返回值	类型	说明
0	int	成功
-1	int	失败