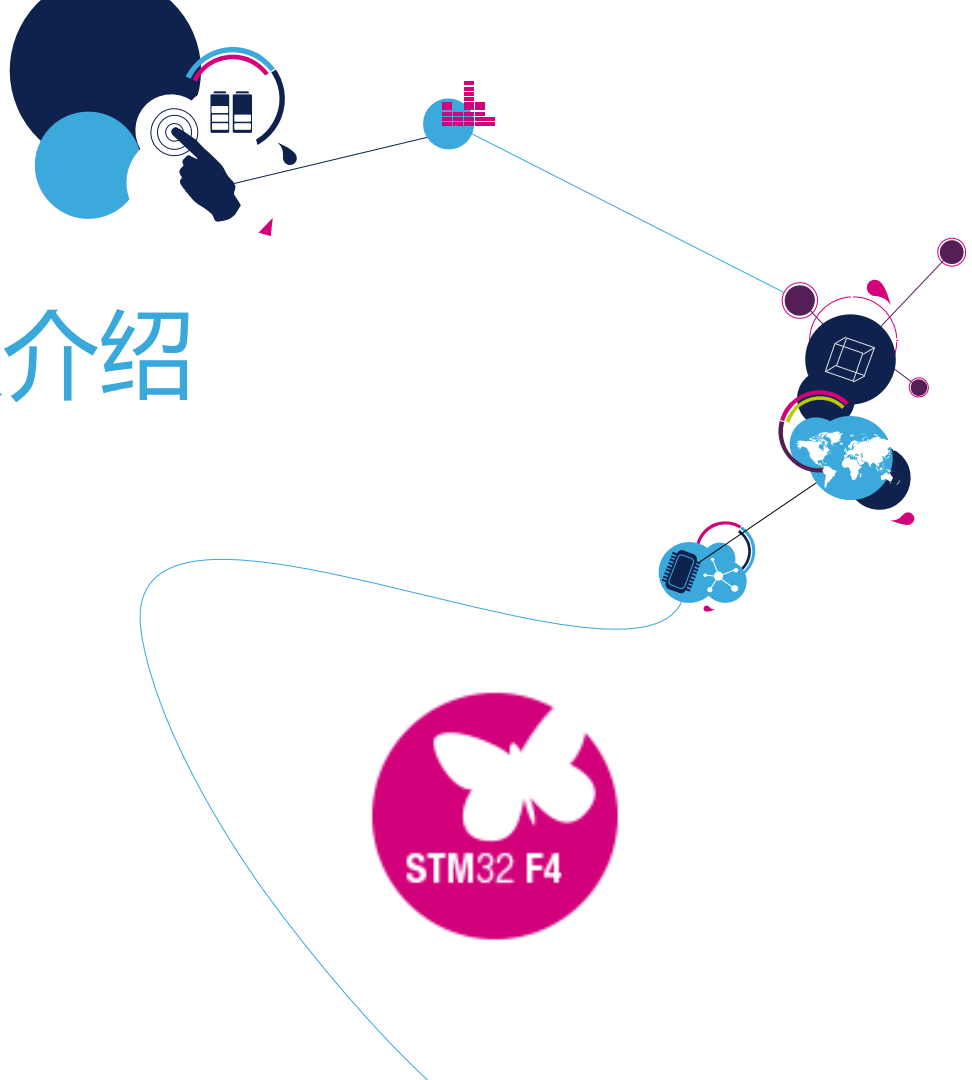
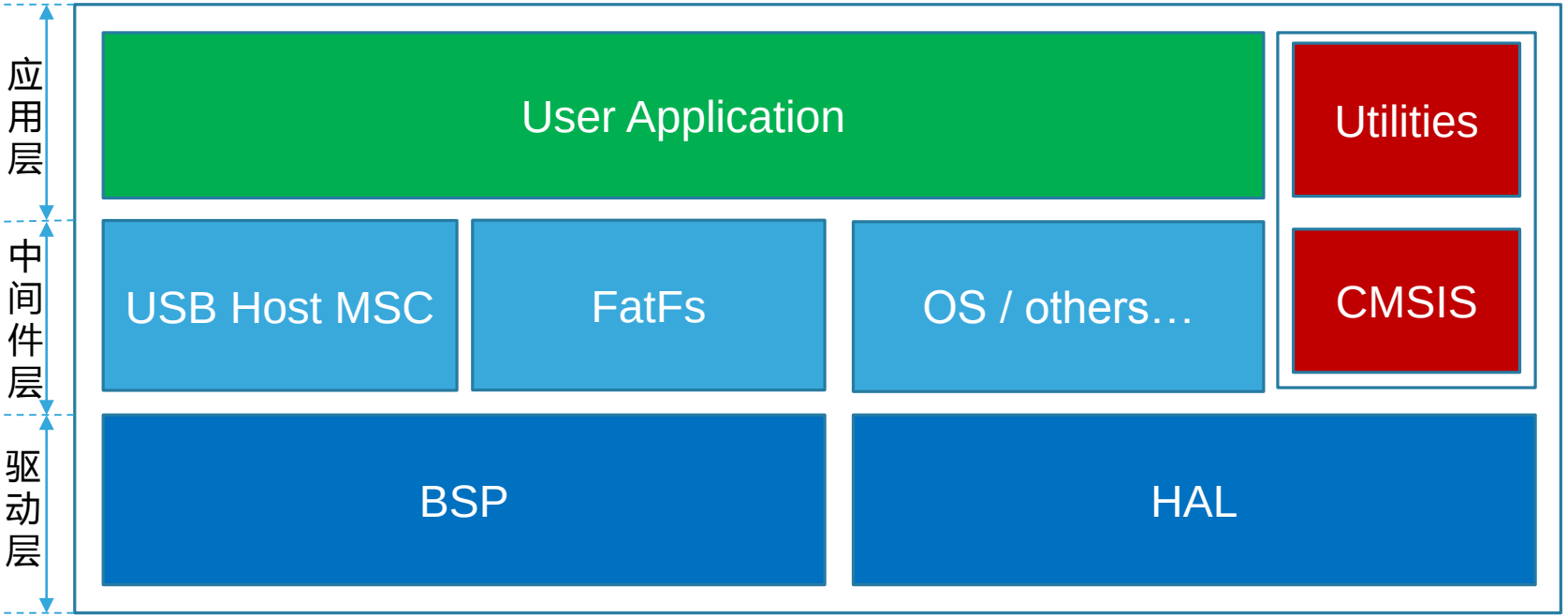


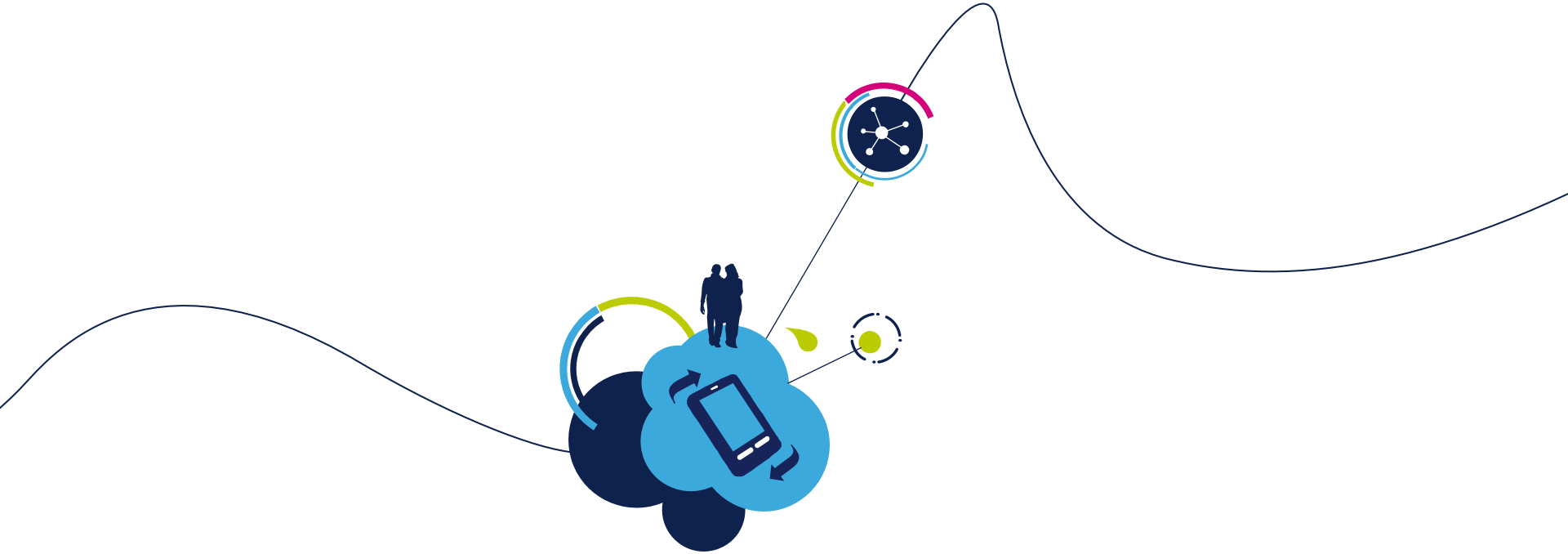
USB Host MSC类介绍

2018年5月

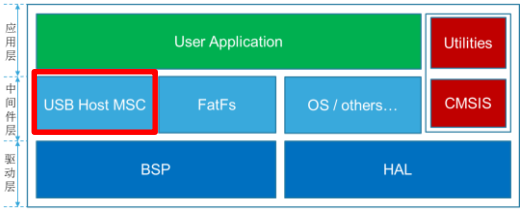
life.augmented

整体软件框架



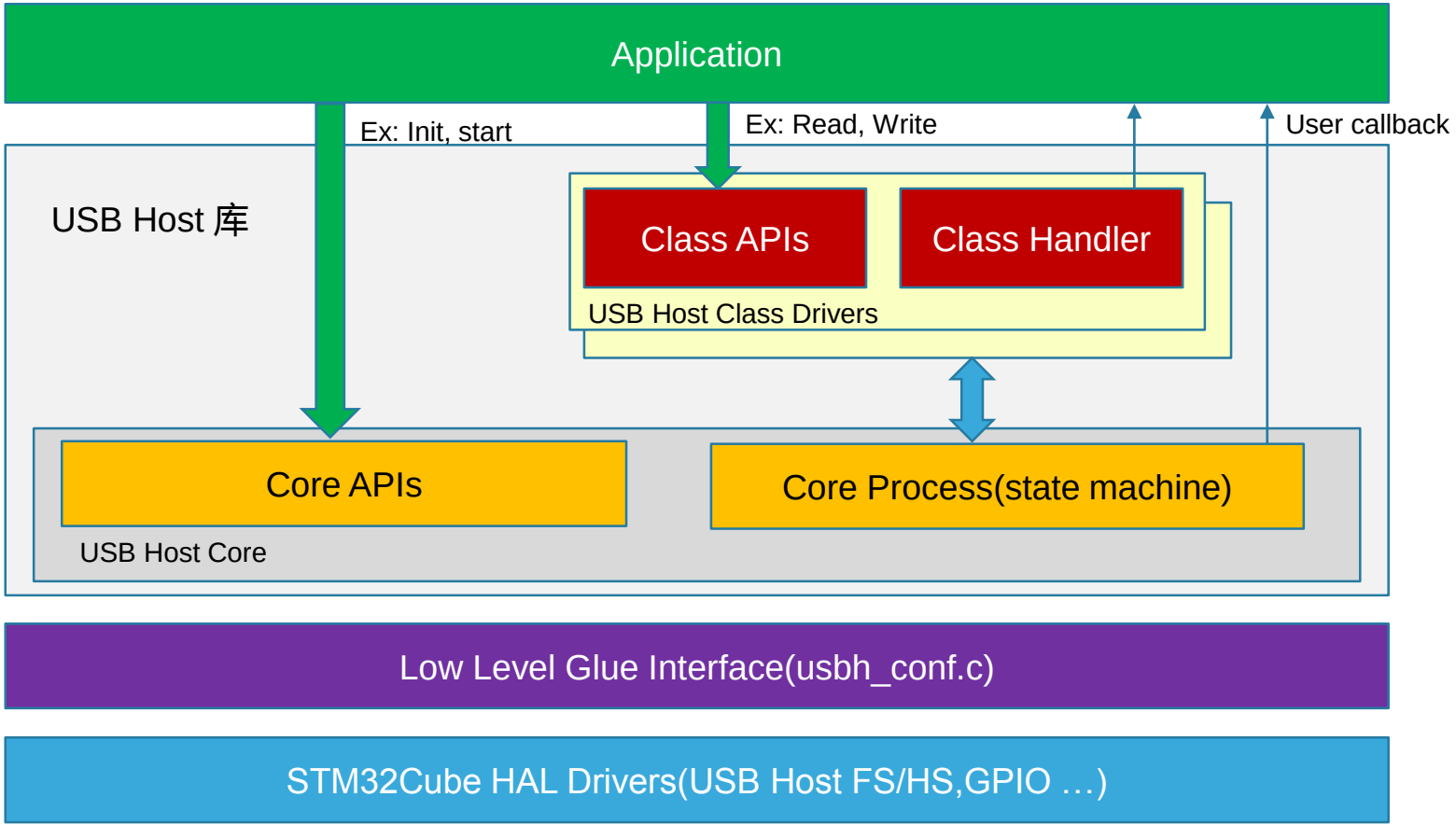


USB Host MSC

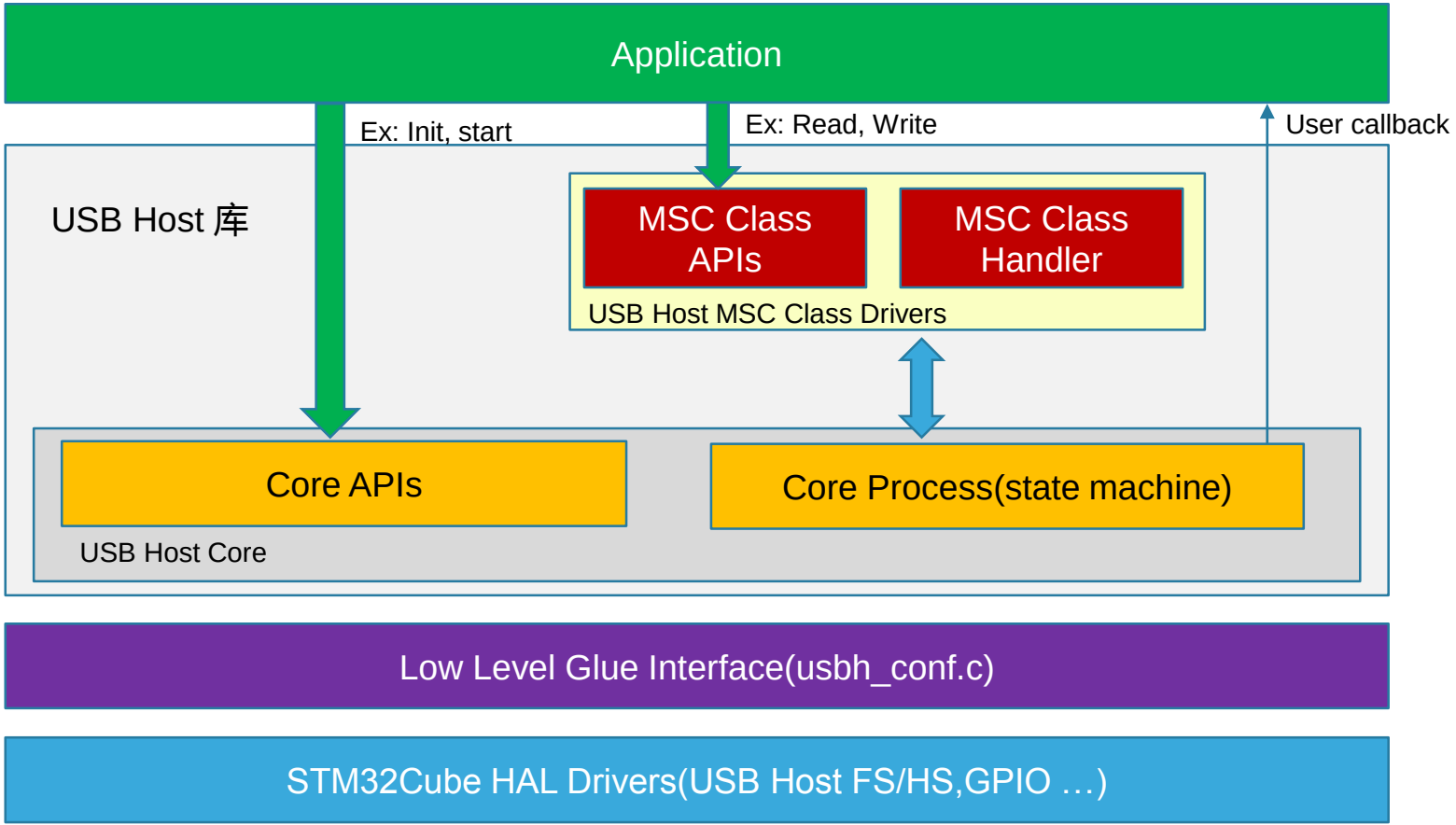


life.augmented

USB Host 架构



USB Host MSC架构



USB Host MSC文件组织

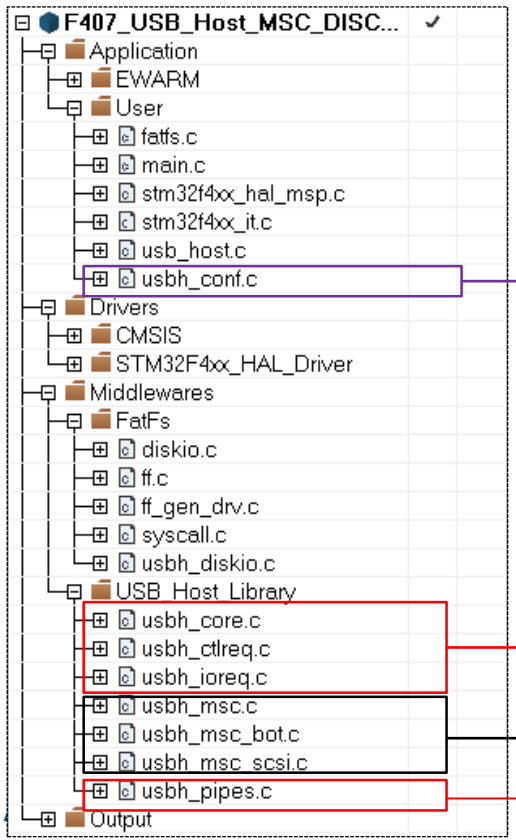
USBH 核心文件

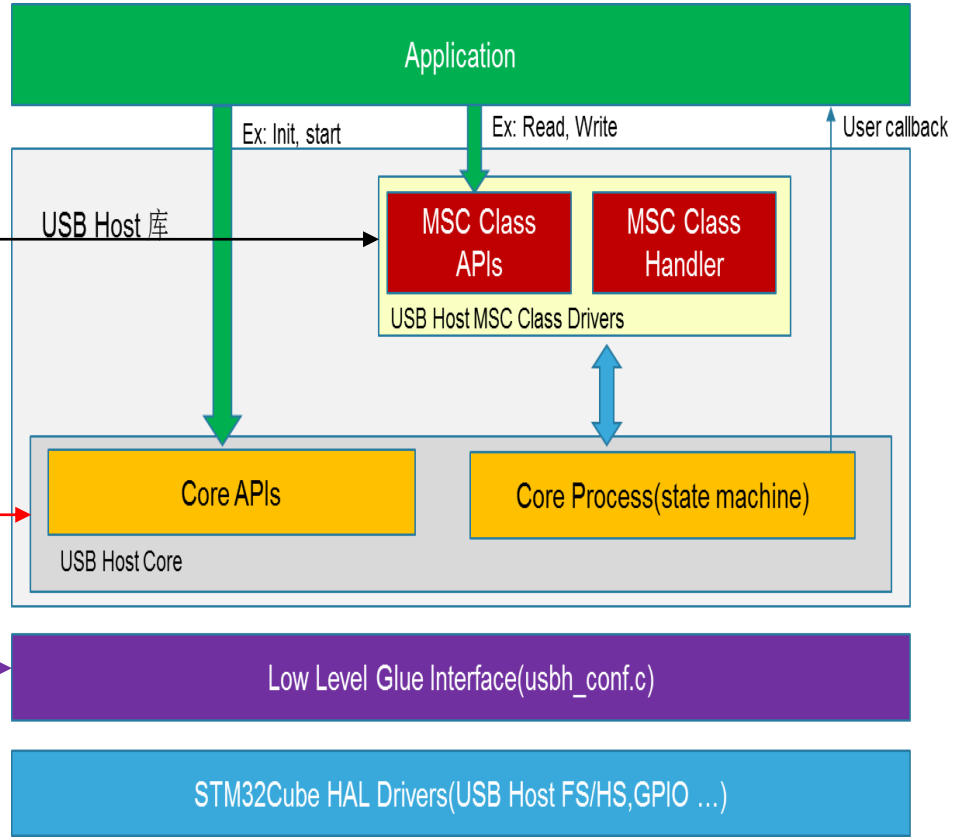
usbh_core.c/.h	状态机，设备检测，枚举，类模块处理
usbh_ctlreq.c/.h	标准控制请求
usbh_ioreq.c/.h	USB传输管理接口
usbh_piples.c/.h	管道控制接口
usbh_conf.c/.h	用户可配置的底层配置文件
usbh_def.h	通用 USB库定义

USBH MSC 类文件

usbh_msc.c	MSC类定义源码文件
usbh_msc_bot.c	Bulk_Only Transfer Protocol BOT传输协议
usbh_msc_scsi.c	scsi指令实现

USB Host MSC 源码文件与架构对应关系





USBH 内核

life.augmented

USBH内核源码文件

USBH 核心文件

usbh_core.c/.h	状态机，设备检测，枚举，类模块处理
usbh_ctlreq.c/.h	标准控制请求
usbh_ioreq.c/.h	USB传输管理接口
usbh_piples.c/.h	管道控制接口
usbh_conf.c/.h	用户可配置的底层配置文件
usbh_def.h	通用 USB库定义

USBH MSC 类文件

usbh_msc.c	MSC类定义源码文件
usbh_msc_bot.c	Bulk_Only Transfer Protocol BOT传输协议
usbh_msc_scsi.c	scsi指令实现

USBH内核与MSC的接口

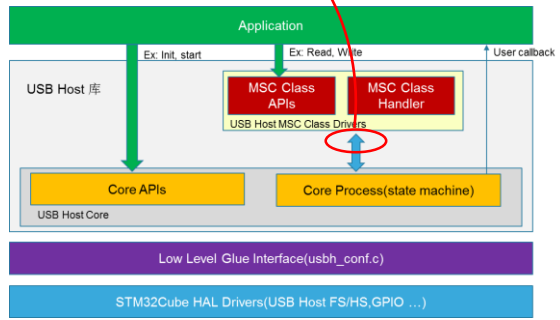

```
typedef struct
{
  const char                *Name;
  uint8_t                   ClassCode;
  USBH_StatusTypeDef  (*Init)       (struct _USBH_HandleTypeDef *phost);
  USBH_StatusTypeDef  (*DeInit)     (struct _USBH_HandleTypeDef *phost);
  USBH_StatusTypeDef  (*Requests)   (struct _USBH_HandleTypeDef *phost);
  USBH_StatusTypeDef  (*BgndProcess) (struct _USBH_HandleTypeDef *phost);
  USBH_StatusTypeDef  (*SOFProcess) (struct _USBH_HandleTypeDef *phost);
  void                      *pData;
} USBH_ClassTypeDef;
```

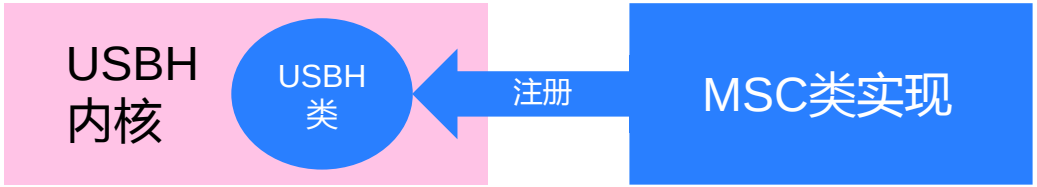
实现

```
USBH_ClassTypeDef  USBH_msc =
{
  "MSC",
  USB_MSC_CLASS,
  USBH_MSC_InterfaceInit,
  USBH_MSC_InterfaceDeInit,
  USBH_MSC_ClassRequest,
  USBH_MSC_Process,
  USBH_MSC_SOFProcess,
  NULL,
};
```

USBH_RegisterClass(&hUsbHostFS, USBH_MSC_CLASS);

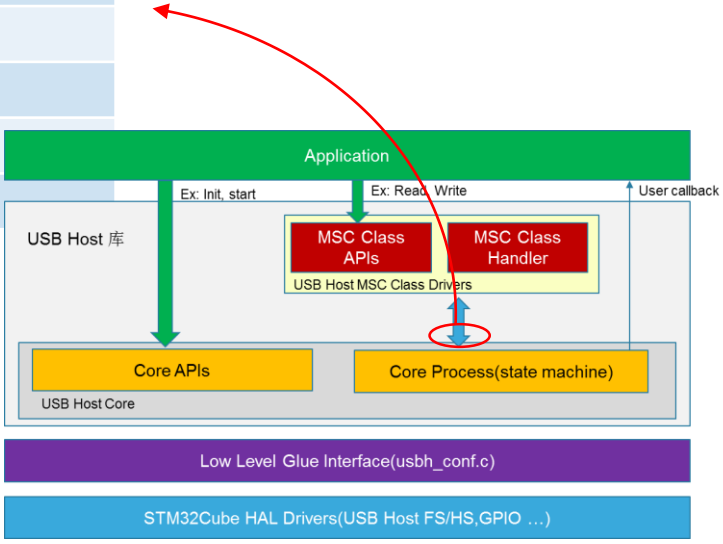
USBH内核回调这些类注册的函数，以此保持两个模块分离。





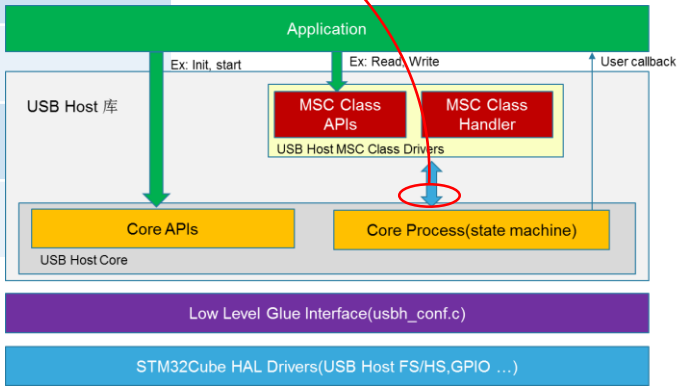
USBH内核对MSC类提供的接口(1/2)

类型	函数	功能描述
IO请求	USBH_BulkReceiveData	bulk管道上数据接收
	USBH_BulkSendData	bulk管道上数据发送
	USBH_CtlReceiveData	控制管道上数据接收
	USBH_CtlSendData	控制管道上数据发送
	USBH_CtlSendSetup	发送一个控制请求
	USBH_InterruptReceiveData	中断管道上数据接收
	USBH_InterruptSendData	中断管道上数据发送
	USBH_IsocReceiveData	同步传输管道上数据接收
	USBH_IsocSendData	同步传输管道上数据发送



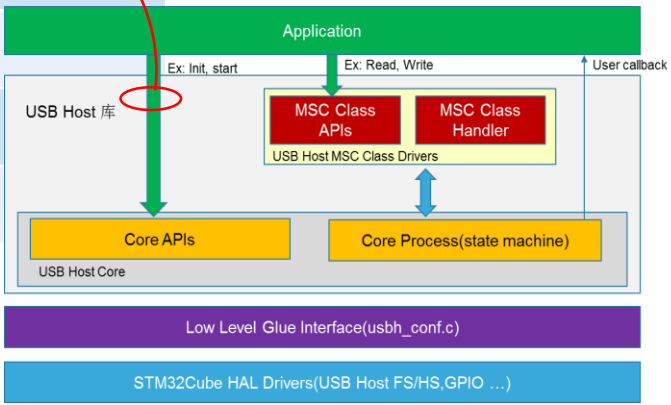
USBH内核对MSC类提供的接口(2/2)

类型	函数	功能描述
管道控制	USBH_OpenPipe	打开管道
	USBH_ClosePipe	关闭管道
	USBH_AllocPipe	为一个管道分配空间
	USBH_FreePipe	释放管道对应的空间
标准控制请求	USBH_GetDescriptor	获取描述符
	USBH_SetInterface	设置接口可选设置
接口工具	USBH_FindInterface	查找接口
	USBH_FindInterfaceIndex	查找接口，返回索引号



USBH内核对应用层提供的接口

API	功能描述
USBH_Init	初始化Host栈和底层模块，在系统启动时调用
USBH_DeInit	反初始化函数
USBH_RegisterClass	向USBH内核注册一个类
USB_ReEnumerate	重新枚举，重新初始化Host栈，VBUS失能后再使能
USBH_Start	使能主机端口VBUS供电和开启底层操作
USBH_Stop	关闭主机端口和VBUS供电并关闭底层操作
USBH_GetActiveClass	返回当前枚举和类请求成功的类
USBH_Process	实现主机状态机管理

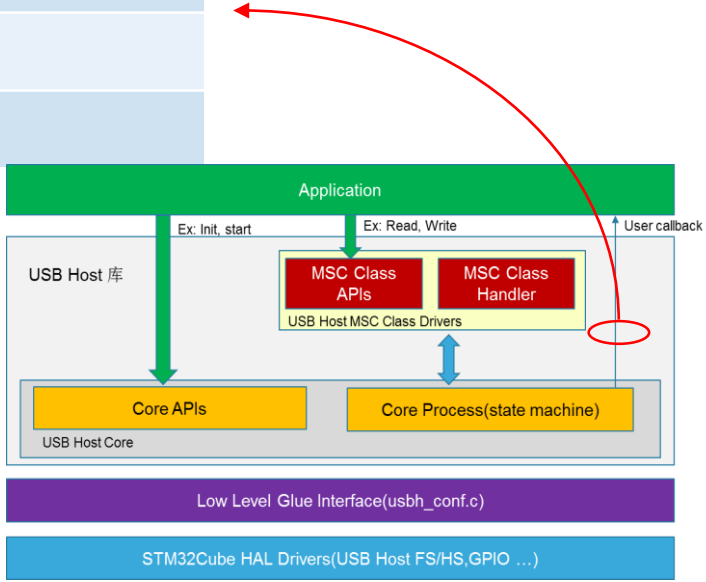


USBH内核对应用层的回调函数

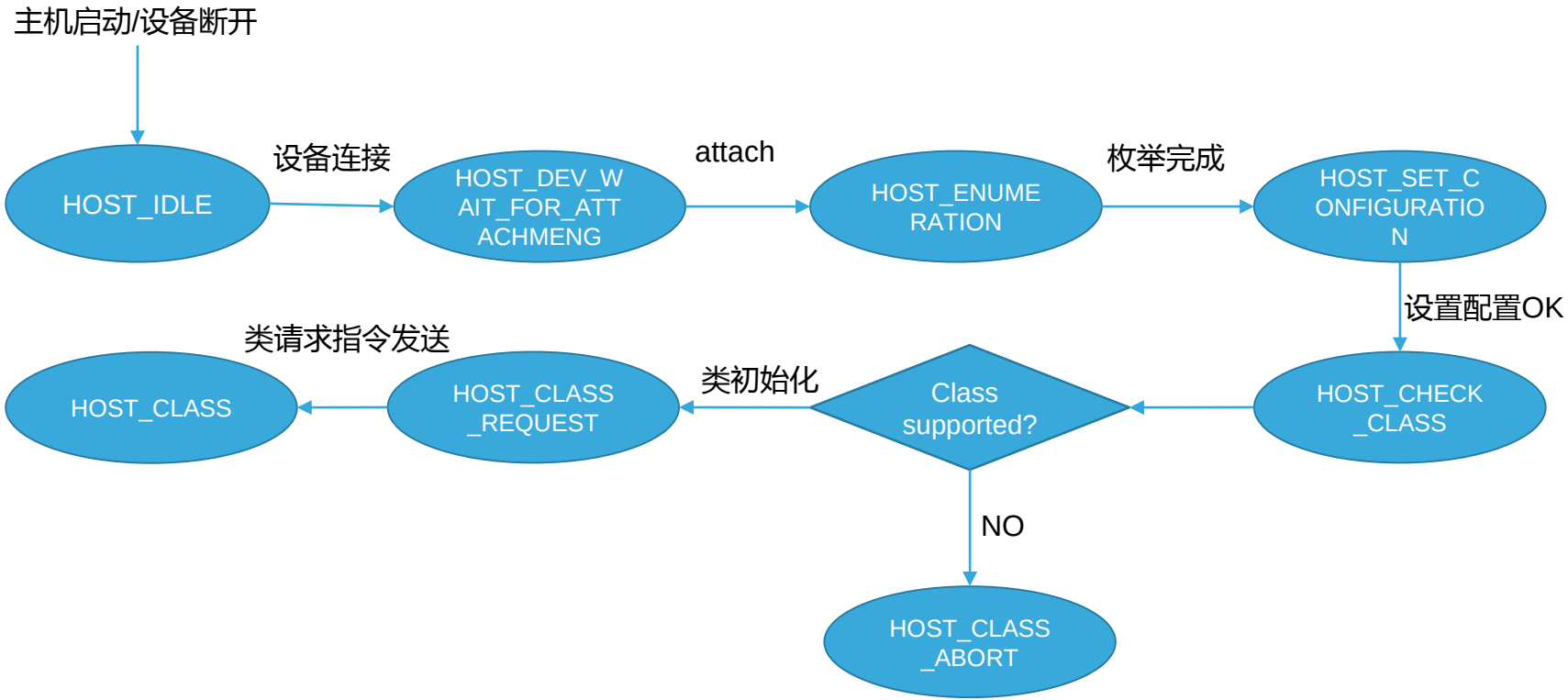
回调事件	描述
HOST_USER_CONNECT	向应用层通知一个USB连接事件
HOST_USER_DISCONNECT	向应用层通知一个USB断开事件
HOST_USER_CLASS_ACTIVE	向应用层通知类请求指令结束事件(当前类被激活)
HOST_USER_SET_CONFIGURATION	向应用层通知标准枚举结束事件
HOST_USER_CLASS_SELECTED	向应用层通知当前类被支持并已被选择

应用层在初始化是通过注册一个USBH_UserProcess函数来接收这些消息并处理

```
USBH_Init(&hUsbHostFS, USBH_UserProcess, HOST_FS);
```



USB Host状态机



USBH MSC类

life.augmented

USBH MSC类源码文件

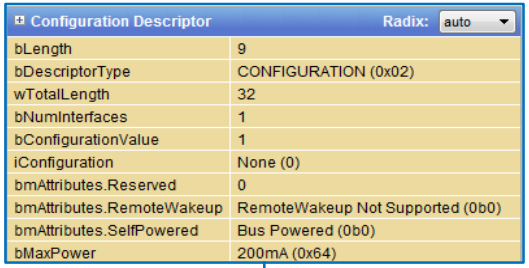
USBH 核心文件

usbh_core.c/.h	状态机，设备检测，枚举，类模块处理
usbh_ctlreq.c/.h	标准控制请求
usbh_ioreq.c/.h	USB传输管理接口
usbh_piples.c/.h	管道控制接口
usbh_conf.c/.h	用户可配置的底层配置文件
usbh_def.h	通用 USB库定义

USBH MSC 类文件

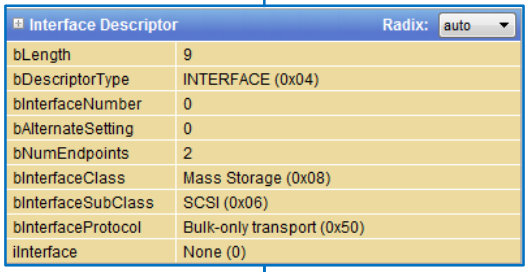
usbh_msc.c	MSC类定义源码文件
usbh_msc_bot.c	Bulk_Only Transfer Protocol BOT传输协议
usbh_msc_scsi.c	scsi指令实现

USB MSC设备的配置描述符结构



Configuration Descriptor	Radix: auto
bLength	9
bDescriptorType	CONFIGURATION (0x02)
wTotalLength	32
bNumInterfaces	1
bConfigurationValue	1
iConfiguration	None (0)
bmAttributes.Reserved	0
bmAttributes.RemoteWakeup	RemoteWakeup Not Supported (0b0)
bmAttributes.SelfPowered	Bus Powered (0b0)
bMaxPower	200mA (0x64)

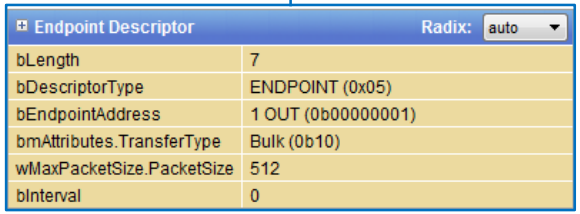
配置描述符



Interface Descriptor	Radix: auto
bLength	9
bDescriptorType	INTERFACE (0x04)
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	2
bInterfaceClass	Mass Storage (0x08)
bInterfaceSubClass	SCSI (0x06)
bInterfaceProtocol	Bulk-only transport (0x50)
iInterface	None (0)

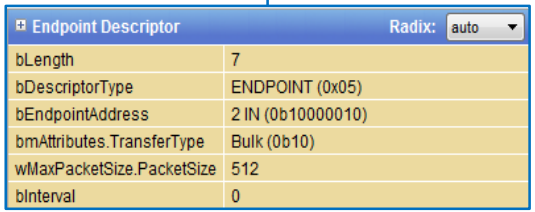
接口描述符

端点1描述符



Endpoint Descriptor	Radix: auto
bLength	7
bDescriptorType	ENDPOINT (0x05)
bEndpointAddress	1 OUT (0b00000001)
bmAttributes.TransferType	Bulk (0b10)
wMaxPacketSize.PacketSize	512
bInterval	0

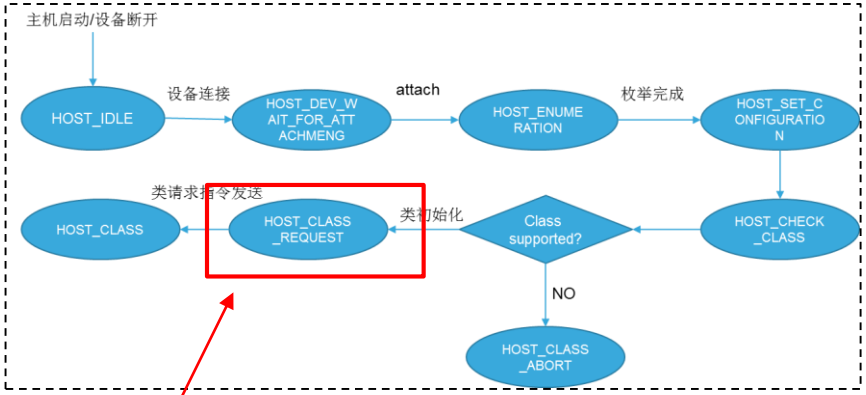
端点2描述符



Endpoint Descriptor	Radix: auto
bLength	7
bDescriptorType	ENDPOINT (0x05)
bEndpointAddress	2 IN (0b10000010)
bmAttributes.TransferType	Bulk (0b10)
wMaxPacketSize.PacketSize	512
bInterval	0

USBH MSC类请求指令

类请求指令	描述
Get_Max_LUN	获取当前最大逻辑单元
BOT_Reset	BOT复位(当USBH内核处于HOST_MSC状态下发生BOT error时)



在标准枚举结束后发送类请求指令

FS	71	0:45.244.000	32 B	01	00	Get Configuration Descriptor	Index=0 Length=32
FS	85	0:45.245.062	2.08 us			[1 SOF]	[Frame: 104]
FS	86	0:45.244.351	16 B	01	00	Get String Descriptor	Index=1 Length=255
FS	100	0:45.246.062	1.00 ms			[2 SOF]	[Frames: 105 - 106]
FS	101	0:45.245.686	26 B	01	00	Get String Descriptor	Index=2 Length=255
FS	115	0:45.248.062	2.08 us			[1 SOF]	[Frame: 107]
FS	116	0:45.247.140	18 B	01	00	Get String Descriptor	Index=3 Length=255
FS	130	0:45.248.498	0 B	01	00	Set Configuration	Configuration=1
FS	140	0:45.248.585	1 B	01	00	Get Max LUN	Max LUN = 0
FS	154	0:45.249.062	1.00 ms			[2 SOF]	[Frames: 108 - 109]
FS	155	0:45.248.679	36 B	01	01	Inquiry [0]	Passed
FS	173	0:45.250.209		01	01	Test Unit Ready [0]	Failed
FS	185	0:45.250.368	14 B	01	01	Request Sense [0]	Sense Key = Unit Attention (Passed)
FS	203	0:45.250.560		01	01	Test Unit Ready [0]	Passed
FS	215	0:45.250.700	8 B	01	01	Read Capacity [0]	Passed

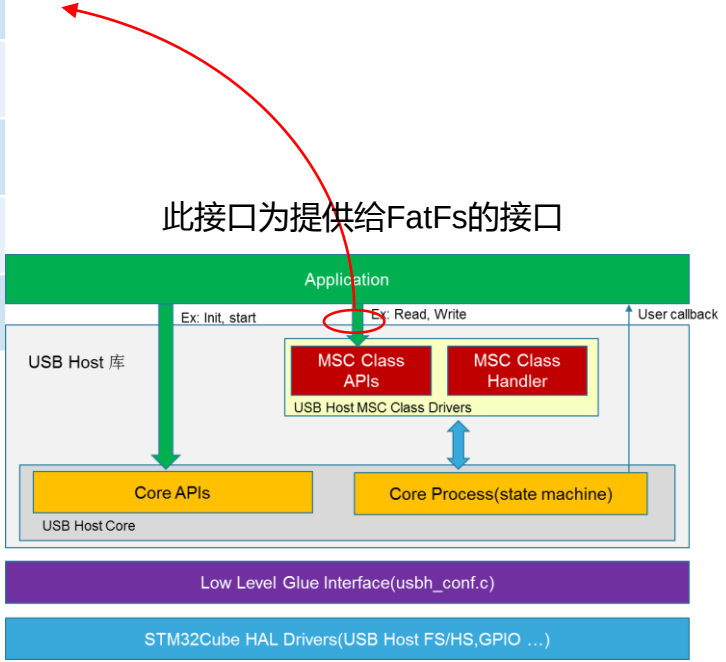
MSC_Trace1.tdc

MSC类对上提供的接口

API	描述
USBH_MSC_Read	从逻辑单元中读取数个扇区(阻塞函数)
USBH_MSC_Write	从逻辑单元中写数个扇区(阻塞函数)
USBH_MSC_GetMaxLUN	获取逻辑单元的数量
USBH_MSC_GetLUNInfo	获取逻辑单元的信息
USBH_MSC_IsReady	检查存储设备是否已经处于读写准备状态

usbh_msc.c

此接口为提供给FatFs的接口



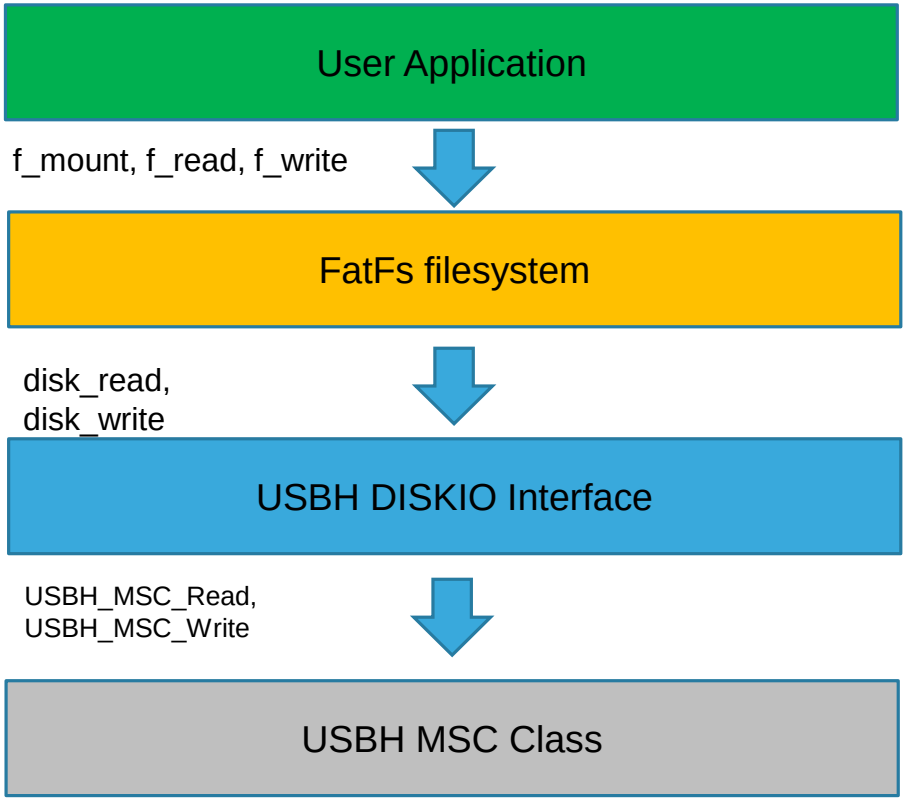
SCSI指令

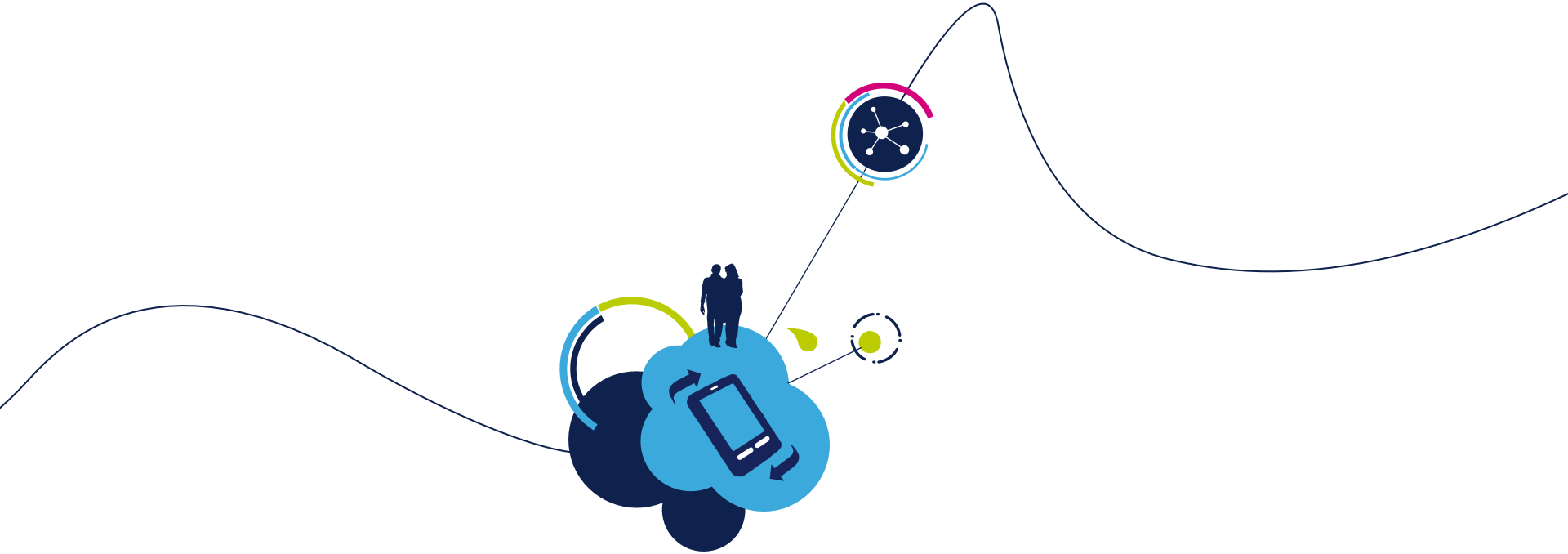
SCSI指令	描述
TestUnitReady	测试媒介是否处于准备状态
ReadCapacity10	读取媒介容量
Inquiry	获取存储设备相关信息，如制作商，版本等
RequestSense	用来获取错误信息
Write10	写一个数据块
Read10	读取一个数据块

usbh_msc_scsi.c

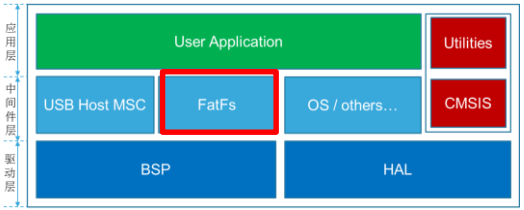
FS	155	0:45.248.679	36 B	01	01		Inquiry [0]	Passed
FS	173	0:45.250.209		01	01		Test Unit Ready [0]	Failed
FS	185	0:45.250.368	14 B	01	01		Request Sense [0]	Sense Key = Unit Attention (Passed)
FS	203	0:45.250.560		01	01		Test Unit Ready [0]	Passed
FS	215	0:45.250.700	8 B	01	01		Read Capacity [0]	Passed
FS	233	0:45.251.062	1.99 s				[2000 SOF]	[Frames: 110 - 61] [Periodic Timeout]
FS	234	0:47.251.232	1.99 s				[2000 SOF]	[Frames: 62 - 13] [Periodic Timeout]
FS	235	0:49.251.402	1.99 s				[2000 SOF]	[Frames: 14 - 2013] [Periodic Timeout]
FS	236	0:51.251.572	330 ms				[331 SOF]	[Frames: 2014 - 296]
FS	237	0:51.580.070	512 B	01	01		Read [0]	LBA = 0 Length = 1 block (Passed)
FS	283	0:51.582.600	3.00 ms				[4 SOF]	[Frames: 297 - 300]
FS	284	0:51.582.036	512 B	01	01		Read [0]	LBA = 1449728 Length = 1 block (Passed)
FS	330	0:51.586.600	2.08 us				[1 SOF]	[Frame: 301]
FS	331	0:51.586.165	512 B	01	01		Read [0]	LBA = 1449729 Length = 1 block (Passed)
FS	377	0:51.587.600	2.08 us				[1 SOF]	[Frame: 302]
FS	378	0:51.587.361	512 B	01	01		Read [0]	LBA = 1466112 Length = 1 block (Passed)
FS	424	0:51.588.600	7.00 ms				[8 SOF]	[Frames: 303 - 310]
FS	425	0:51.588.541	512 B	01	01		Write [0]	LBA = 1466112 Length = 1 block (Passed)

使用USBH MSC类





FatFs文件系统

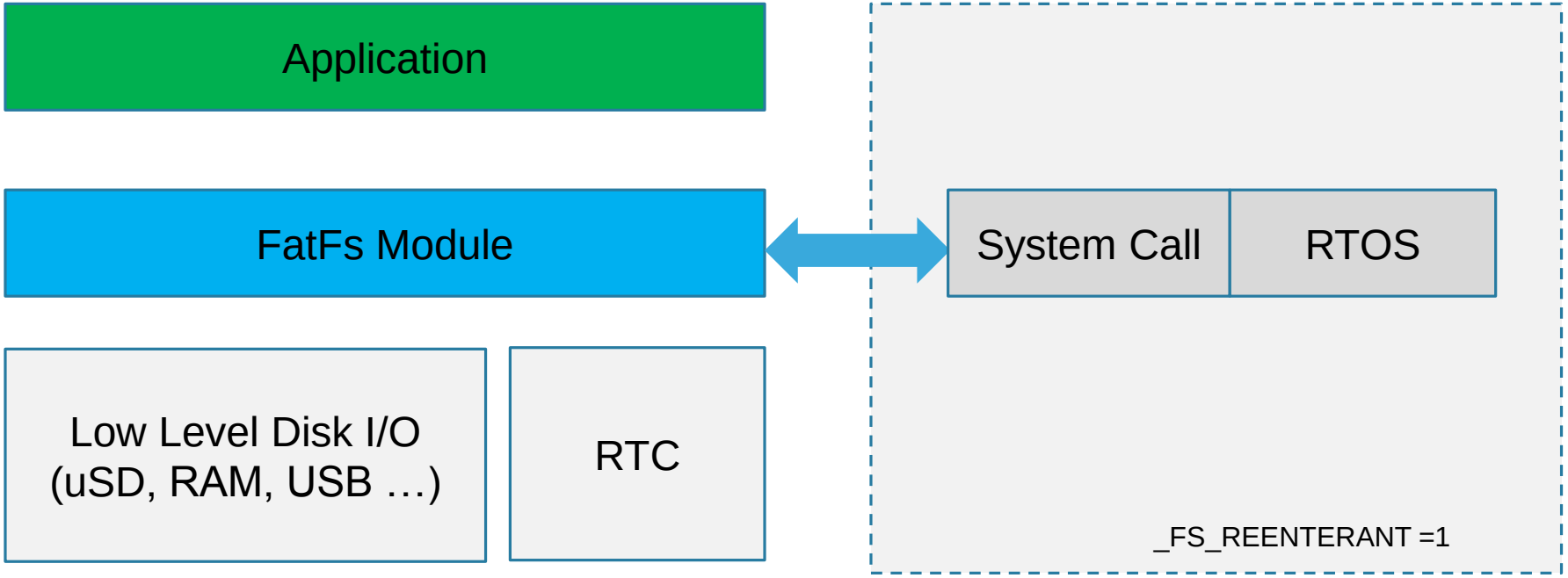


life.augmented

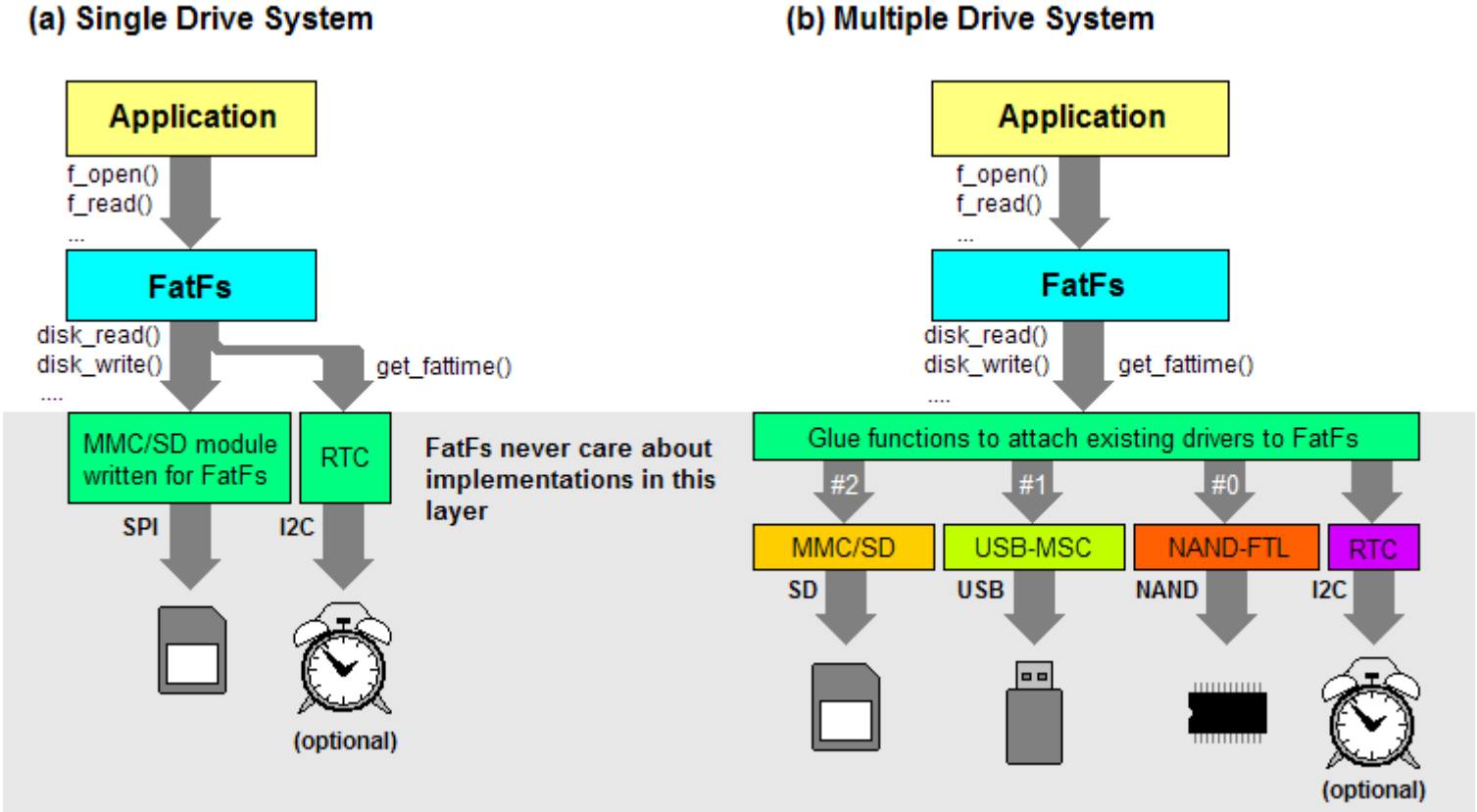
FatFs概述

- FatFs为一开源项目，兼容FAT文件系统，适用于小型嵌入式系统。
- 完全与硬件层分离，适合移植应用于各种存储设备。
- 占用FLASH，RAM空间小。
- 灵活配置，利于裁剪优化模块。
- 支持多个卷(<=10)，长文件名，RTOS，只读模式，FAT12/16/32。
- 同时打开的文件数无限制，取决于当前内存。
- 文件大小，取决于FAT规格，最大4G-1 bytes。
- 卷大小，取决于FAT规格，最大2T(512B/sector)。
- 簇大小，取决于FAT规格，最大64K(512B/sector)。
- 扇区大小，取决于FAT规格，最大4K。

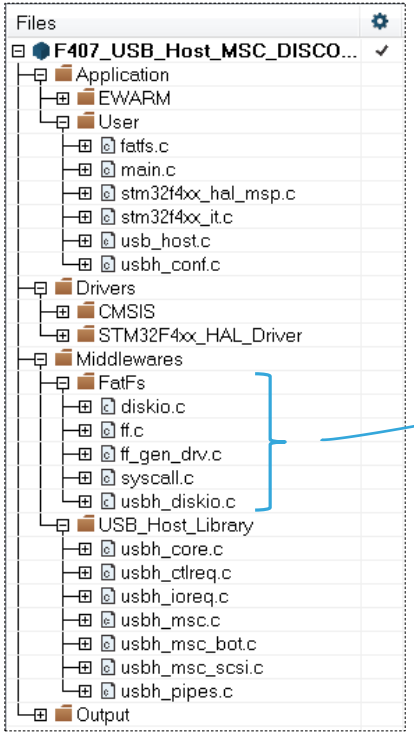
FatFs系统框图

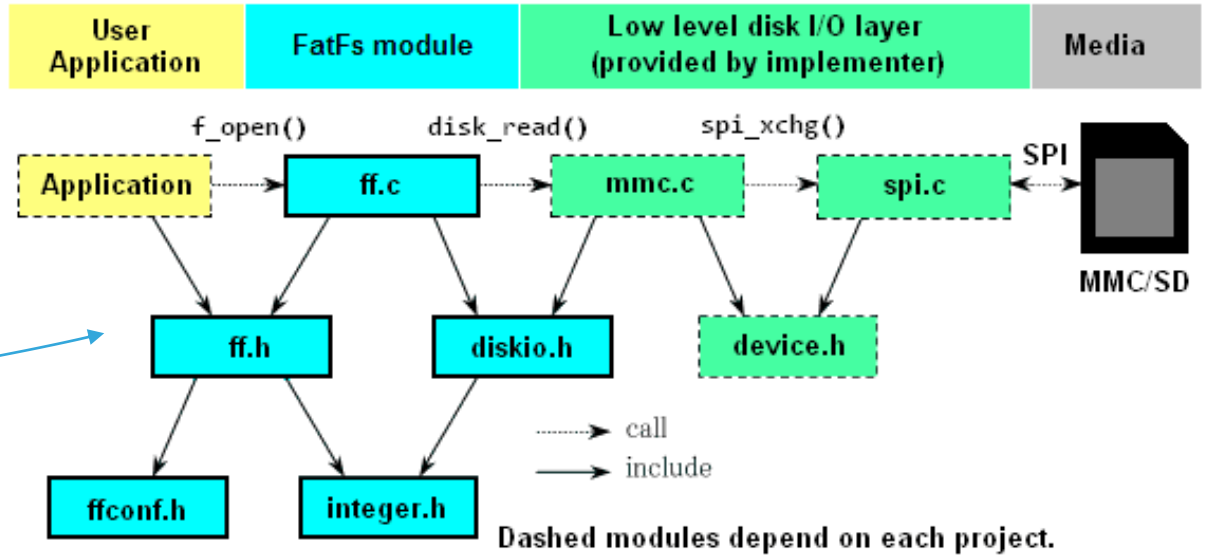


Single Drive System *VS* Multiple Drive System



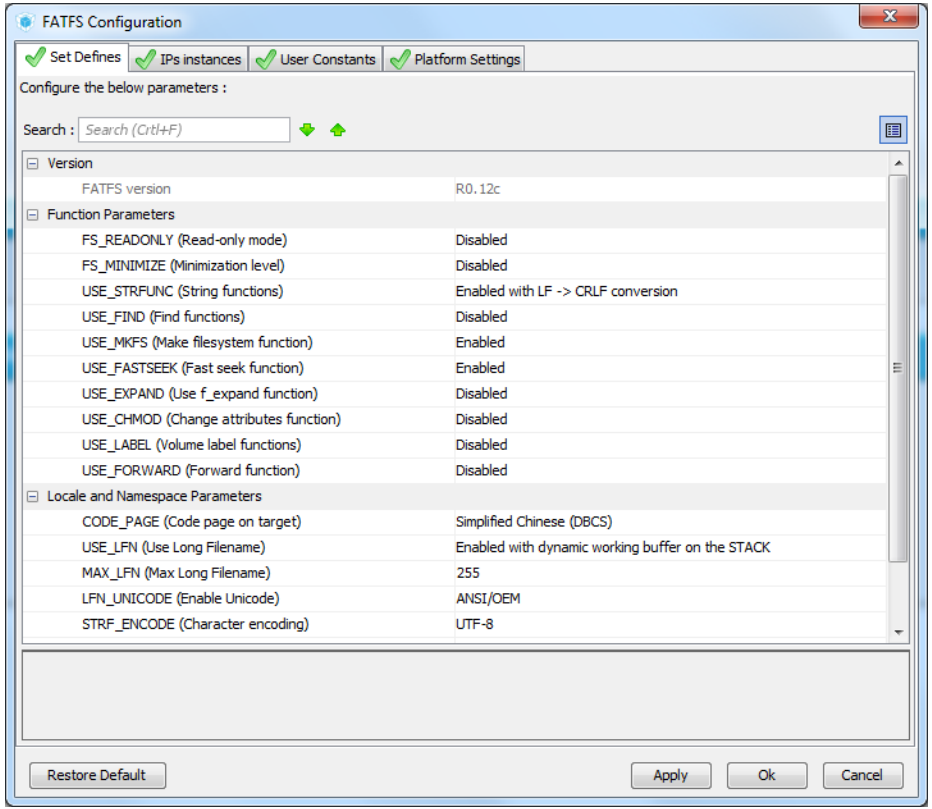
FatFs源文件组织





life.augmented

FatFs配置



- Function Configurations
  - FF_FS_READONLY
  - FF_FS_MINIMIZE
  - FF_USE_STRFUNC
  - FF_USE_FIND
  - FF_USE_MKFS
  - FF_USE_FASTSEEK
  - FF_USE_EXPAND
  - FF_USE_CHMOD
  - FF_USE_LABEL
  - FF_USE_FORWARD
- Namespace and Locale Configurations
  - FF_CODE_PAGE
  - FF_USE_LFN
  - FF_MAX_LFN
  - FF_LFN_UNICODE
  - FF_LFN_BUF. FF_SFN_BUF
  - FF_STRF_ENCODE
  - FF_FS_RPATH
- Volume/Drive Configurations
  - FF_VOLUMES
  - FF_STR_VOLUME_ID
  - FF_VOLUME_STRS
  - FF_MULTI_PARTITION
  - FF_MIN_SS. FF_MAX_SS
  - FF_USE_TRIM
  - FF_FS_NOFSINFO
- System Configurations
  - FF_FS_TINY
  - FF_FS_EXFAT
  - FF_FS_NORTC
  - FF_NORTC_MON. FF_NORTC_MDAY. FF_NORTC_YEAR
  - FF_FS_LOCK
  - FF_FS_REENTRANT
  - FF_FS_TIMEOUT
  - FF_SYNC_t

FatFs APIs

文件访问

- f_open - Open/Create a file
- f_close - Close an open file
- f_read - Read data from the file
- f_write - Write data to the file
- f_lseek - Move read/write pointer, Expand size
- f_truncate - Truncate file size
- f_sync - Flush cached data
- f_forward - Forward data to the stream
- f_expand - Allocate a contiguous block to the file
- f_gets - Read a string
- f_putc - Write a character
- f_puts - Write a string
- f_printf - Write a formatted string
- f_tell - Get current read/write pointer
- f_eof - Test for end-of-file
- f_size - Get size
- f_error - Test for an error

文件夹访问

- f_opendir - Open a directory
- f_closedir - Close an open directory
- f_readdir - Read an directory item
- f_findfirst - Open a directory and read the first item matched
- f_findnext - Read a next item matched

文件和文件夹管理

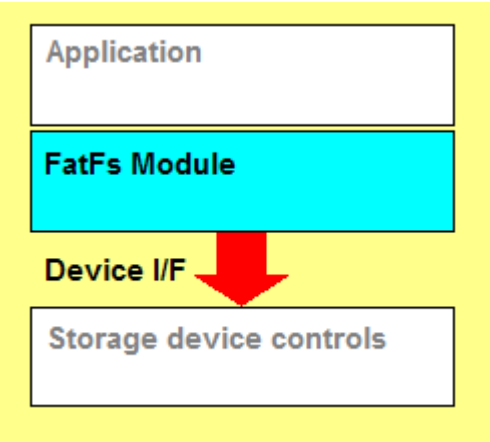
- f_stat - Check existance of a file or sub-directory
- f_unlink - Remove a file or sub-directory
- f_rename - Rename/Move a file or sub-directory
- f_chmod - Change attribute of a file or sub-directory
- f_utime - Change timestamp of a file or sub-directory
- f_mkdir - Create a sub-directory
- f_chdir - Change current directory
- f_chdrive - Change current drive
- f_getcwd - Retrieve the current directory and drive

卷管理和系统配置

- f_mount - Register/Unregister the work area of the volume
- f_mkfs - Create an FAT volume on the logical drive
- f_fdisk - Create logical drives on the physical drive
- f_getfree - Get total size and free size on the volume
- f_getlabel - Get volume label
- f_setlabel - Set volume label
- f_setcp - Set active code page

Media Access Interface

- disk_status - Get device status
- disk_initialize - Initialize device
- disk_read - Read sector(s)
- disk_write - Write sector(s)
- disk_ioctl - Control device dependent functions
- get_fattime - Get current time



- FatFs模块需要使用到一些底层函数，需要由用户根据媒介类型提供这些函数的具体实现。
- 在CubeMx中，这些函数可以自动生成,使之与底层特定媒介对应API无缝连接(如:FatFs+U盘, FatFs+SD卡)。

FatFs对接USB HOST MSC


```
const Diskio_drvTypeDef  USBH_Driver =
{
  USBH_initialize,
  USBH_status,
  USBH_read,
#if  _USE_WRITE == 1
  USBH_write,
#endif /* _USE_WRITE == 1 */
#if  _USE_IOCTL == 1
  USBH_ioctl,
#endif /* _USE_IOCTL == 1 */
};
```

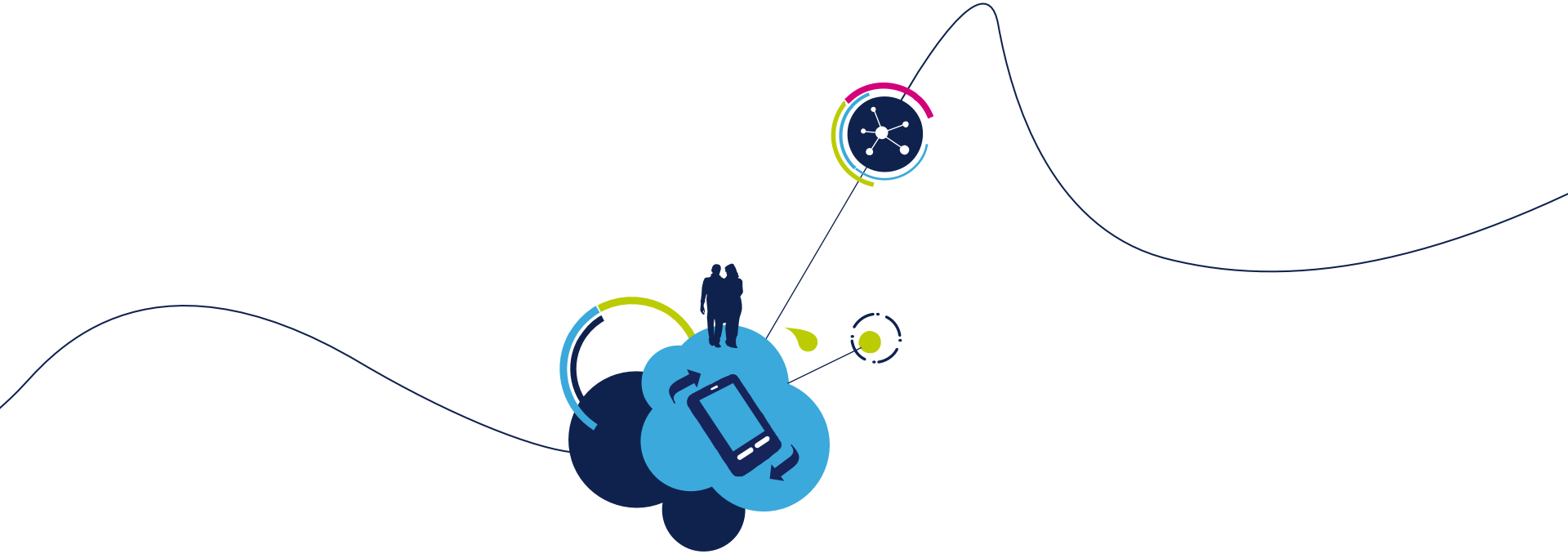
用户实现的FatFs的MAI驱动, 这里CubeMX可以自动生成

将此驱动注册到FatFs模块中

```
void MX_FATFS_Init(void)
{
retUSBH = FATFS_LinkDriver(&USBH_Driver, USBH_Path);
}
```

```
uint8_t FATFS_LinkDriverEx(Diskio_drvTypeDef *drv, char *path, uint8_t lun)
{
  uint8_t ret = 1;
  uint8_t DiskNum = 0;
  if(disk.nbr <= _VOLUMES)
  {
    disk.is_initialized[disk.nbr] = 0;
    disk.drv[disk.nbr] = drv;
    disk.lun[disk.nbr] = lun;
    DiskNum = disk.nbr++;
    path[0] = DiskNum + '0';
    path[1] = ':';
    path[2] = '/';
    path[3] = 0;
    ret = 0;
  }
  return ret;
}
```

注册的驱动存储在这，可以存多个，也就是说，可以同时支持多个媒介。



高级话题

life.augmented

FatFs裁剪(1/2)

FatFs模块所占存储容量取决于用户如何配置，假如编译器为GCC，FatFs R0.13a版本为例:

- **所占FLASH大小**:

  FF_FS_READONLY =0 && FF_FS_MINIMIZE =0 => 6.3K

  FF_FS_READONLY =0 && FF_FS_MINIMIZE =3 => 4.4K

  FF_FS_READONLY =1 && FF_FS_MINIMIZE =0 => 2.8K

  FF_FS_READONLY =1 && FF_FS_MINIMIZE =3 => 2.2K

- **BSS大小**: V*4 +2 (V为挂载的卷数量)

- **运行时内存**:

  FF_FS_TINY =0 => V*564+F*552 (V为挂载的卷数量,F为打开的文件数量)

  FF_FS_TINY =1 => V*564+F*40

其他选项配置均采用默认参数

FatFs裁剪(2/2)

函数是否可用与FatFs配置的对应关系如右边表格所示:

Function	FF_FS_MINIMIZE				FF_FS_READONLY		FF_USE_STRFUNC		FF_FS_RPATH			FF_USE_FIND		FF_USE_CHMOD		FF_USE_EXPAND		FF_USE_LABEL		FF_USE_MKFS		FF_USE_FORWARD		FF_MULTI_PARTITION	
	0	1	2	3	0	1	0	1	0	1	2	0	1	0	1	0	1	0	1	0	1	0	1	0	1
f_mount																									
f_open																									
f_close																									
f_read																									
f_write						x																			
f_sync						x																			
f_lseek				x																					
f_opendir			x	x																					
f_closedir			x	x																					
f_readdir			x	x																					
f_findfirst			x	x								x													
f_findnext			x	x								x													
f_stat		x	x	x																					
f_getfree		x	x	x		x																			
f_truncate		x	x	x		x																			
f_unlink		x	x	x		x																			
f_mkdir		x	x	x		x																			
f_rename		x	x	x		x																			
f_chdir									x																
f_chdrive									x																
f_getcwd									x	x															
f_chmod						x								x											
f_utime						x								x											
f_getlabel																		x							
f_setlabel						x												x							
f_expand						x										x									
f_forward																						x			
f_mkfs						x														x					
f_fdisk						x														x				x	
f_putc						x	x																		
f_puts						x	x																		
f_printf						x	x																		
f_gets							x																		

U盘兼容性讨论

- 案例1

  设备不支持BOT_RESET类指令(回复STALL)

  ->主机需要发送Set Port Feature(PORT_RESET)来复位。

- 案例2

  设备不支持Get_Max_Lun类指令(回复STALL)

  ->主机发送Clear_Feature(EP_Halt),并认为该 U 盘仅包含一个逻辑盘。

这些 U 盘并没有完全遵守大容量规范!

Thank you

/STM32

@ST_World

st.com/e2e

life.augmented