Fasal Flasher

1.0

Generated by Doxygen 1.9.7

Chapter 1

Fasal Flasher

This repository contains Fasal Flasher Source Code. Dive right into the codebase through AppFasal_Run invoked in main.c !

1.0.1 Table of contents

1.0.2 Context

Fasal Flasher is an Internal tool meant as a production floor Aid. Its primary purpose is to update the contents flash IC W25Qxx either with the contents of the Fasal Flasher onboard SD-Card or over a serial terminal over XModem protocol. The device has the following salient features

1. Powered by USB either through PC, Mobile phone or any USB source
2. File Input from either Serial console through USB connector or SD-Card
3. Slide switch to select between file source of SD-Card and Serial console (X-modem)
4. The W25Qxx flash IC in external board is powered through the Fasal Flasher and programmed over SPI
5. PushButton to trigger the transfer in the configured Mode
6. Tri-Color LED for visual indication about the process result
7. Logs over the same USB cable with information about the running application
8. File Integrity check via CRC in case of SD-Card transfer mode
9. File Integrity check inbuilt via XModem protocol in case of XModem transfer mode

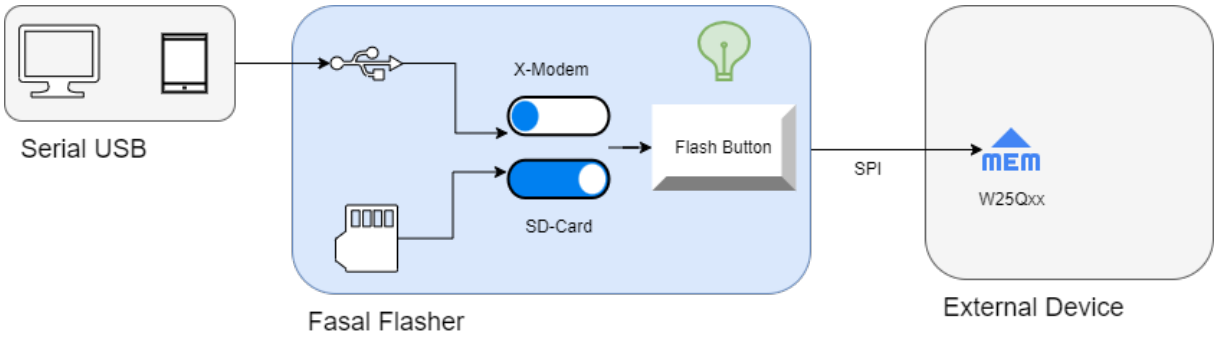


Figure 1.1 Fasal Flasher Usage

1.0.3 Project-structure

1. The project is divided into the following folders
2. SourceCode/FasalFlasher/Core : Contains CubeMX Generated Files utilizing the STM HAL. These files are mostly unmodified and used as is
3. SourceCode/FasalFlasher/Drivers : STM HAL files
4. SourceCode/FasalFlasher/FATFS : FatFS library
5. SourceCode/FasalFlasher/Middlewares : Contains STM crypto library and FatFS library, used as is without any modifications
6. SourceCode/FasalFlasher/User_Files : All custom code for the Fasal Application is present in this folder
   (a) SourceCode/FasalFlasher/User_Files/AppCommon : Contains modules used across the whole application, these include
      i. SourceCode/FasalFlasher/User_Files/AppCommon/AppConfiguration : Project version and compile time configuration constants
      ii. SourceCode/FasalFlasher/User_Files/AppCommon/AppUtility : Utility functions common across all Fasal CodeBases
      iii. SourceCode/FasalFlasher/User_Files/AppCommon/ConfigSetting : Module to check Board HW configuration
      iv. SourceCode/FasalFlasher/User_Files/AppCommon/Console : Console for User logs and X-↩ Modem
      i. SourceCode/FasalFlasher/User_Files/AppCommon/PushButton : PushButton module
      i. SourceCode/FasalFlasher/User_Files/AppCommon/TriColorLED : TriColor LED module

   (b) SourceCode/FasalFlasher/User_Files/AppFasal : Top level application code, Application entry point
   (c) SourceCode/FasalFlasher/User_Files/AppStorage : Top level storage module built atop Flash and SD-card file systems respectively
      i. SourceCode/FasalFlasher/User_Files/AppStorage/AppFlashFS : Filesystem based on LittleFS file system built atop W25Qxx flash IC
      ii. SourceCode/FasalFlasher/User_Files/AppStorage/AppSDFS : Filesystem based on FatFS file system built atop SPI based SD-Card
   (d) SourceCode/FasalFlasher/User_Files/xModem : xModem module built on top of UART based console module

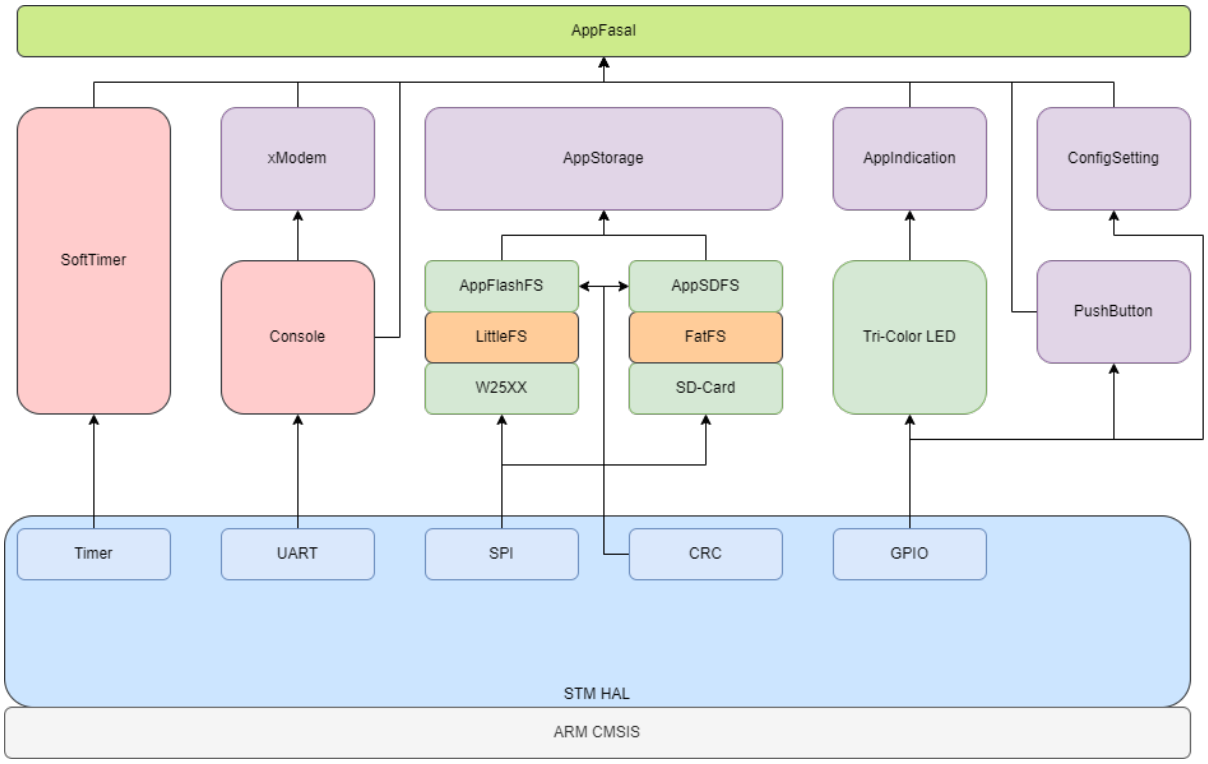


Figure 1.2 Application

```
+---BUILD_DEBUG
+---BUILD_PROD
+---Core
¦   +---Inc
¦   +---Src
¦   +---Startup
+---Drivers
¦   +---CMSIS
¦   ¦   +---Device
¦   ¦   ¦   +---ST
¦   ¦   ¦       +---STM32F1xx
¦   ¦   ¦           +---Include
¦   ¦   ¦           +---Source
¦   ¦   ¦               +---Templates
¦   ¦   +---Include
¦   +---STM32F1xx_HAL_Driver
¦       +---Inc
¦       ¦   +---Legacy
¦       +---Src
+---FATFS
¦   +---App
¦   +---Target
+---Middlewares
¦   +---Third_Party
¦       +---FatFs
¦           +---src
¦               +---option
+---User_Files
    +---AppCommon
    ¦   +---AppConfiguration
    ¦   +---AppUtility
    ¦   ¦   +---AppProfiler
    ¦   ¦   +---SoftTimer
    ¦   +---ConfigSetting
    ¦   +---Console
    ¦   +---PushButton
    ¦   +---TriColorLED
    +---AppFasal
    +---AppStorage
    ¦   +---AppFlashFS
    ¦   ¦   +---LittleFS
    ¦   ¦   +---W25Qxx
    ¦   +---AppSDFS
    +---xModem
```

1.0.4 Code-flow

1. The code flow and the corresponding modules are documented in this section
2. The code starting point is main. All HAL modules are initialized here along with Application level code housed in AppFasal.h
3. AppFasal_Init Initializes the application, sends startup message and sets up the TriColorLED, console and startup timer
4. The function AppFasal_Run is the application state-machine that drives the application
   (a) Module level initialization is carried out in the first step
   (b) The application then waits for the user to press the flash user button
   (c) External flash is then initialized before progressing to the file transfer phase
   (d) Depending on the hardware slide switch setting, the device can then proceed in one of the following modes
      i. SD-Card Transfer Mode:
         A. SD-Card is initialized and it is ensured that the Golden image is present
         B. Golden image is transferred from the SD card to the external flash
         C. CRC of the same file in SD-Card and in the now transferred flash are computed and checked against each other
      ii. XModem Transfer Mode:
         A. File Must be transferred over XModem 1K option though a serial terminal [Baud: 115200, Data: 8b, Stop Bit: 1b]
   (e) In either mode, On successful reception of Golden Image, the Firmware then enters the termination stage, indicates success and waits on the flash user button stage
   (f) In either mode failure of any of the steps prior to successful transfer results in the application state-machine indicating the failure reason and jumping back flash user button stage

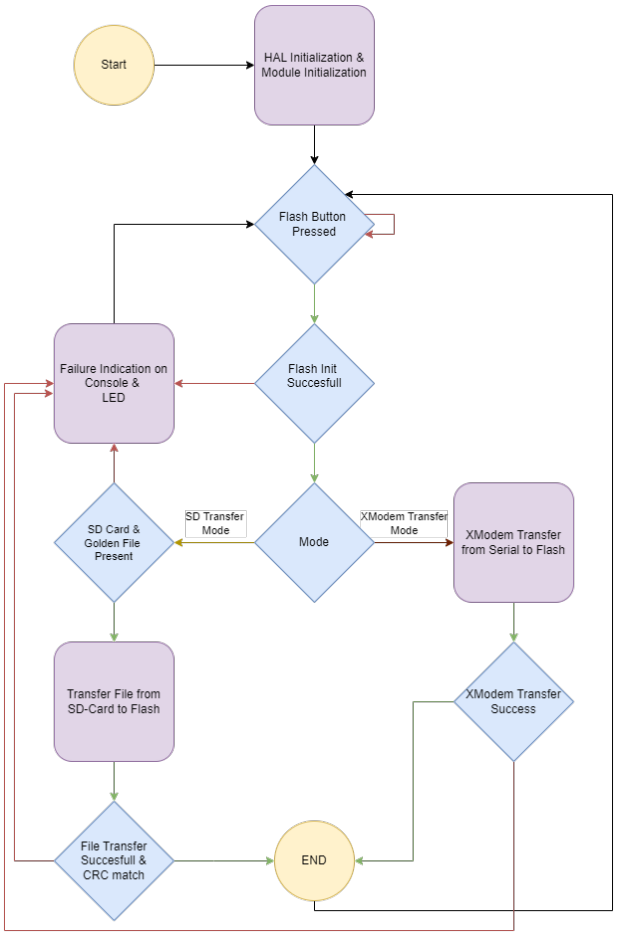


Figure 1.3 APP

1.0.5 Build-Instruction

1. Built using CubeIDE Version: 1.15.1 Build: 21094_20240412_1041 (UTC)
2. Both BUILD_DEBUG and BUILD_PROD Uses Linker script STM32f103RETX_FLASH.ld with flash offset of 0x0800 0000, application size set to 512KB
3. The following build configurations are built into the codebase
   (a) *BUILD_DEBUG* : Debug build used during development with all debug symbols enabled
      - Optimization level : None
      - Symbols defined : DEBUG | STM32F103xE | USE_HAL_DRIVER

   (b) *BUILD_PROD* : Production build for filed use
      - Optimization level : Ofast
      - Symbols defined : NDEBUG | STM32F103xE | USE_HAL_DRIVER

1. SourceCode/FasalFlasher/User_Files/AppCommon/AppConfiguration for changing compile time build features

1.0.6 Docs

1. Doxygen files are generated under Docs/html by running the following command in the current folder `doxygen Doxyfile`
2. The files can be viewed by opening Index.html in any browser. Files located here Docs/html
3. Latex Generated PDF file is preset at Docs/Design_Document/FasalFlasher_FirmwareDesignDocument.pdf

1.0.7 Contact-Me

- `Vishal keshava Murthy` over E-Mail
- `Vishal` on Slack

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 df_dataparam Struct Reference

Data Fields

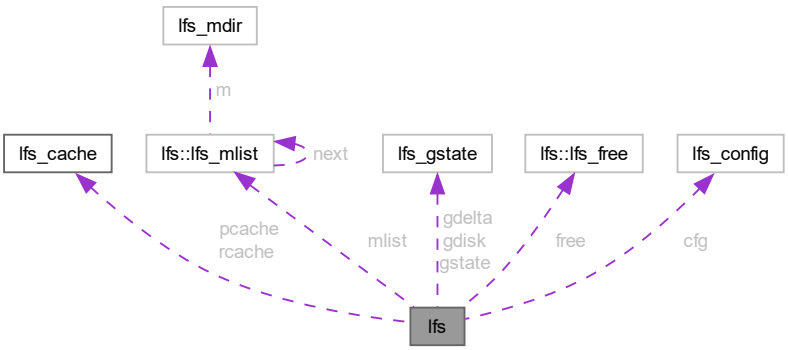
- uint8_t **txbuff** [DF_PAGE_SIZE]
- uint8_t **rxbuff** [DF_PAGE_SIZE]
- uint8_t **txbuff_len**
- uint8_t **rxbuff_len**
- uint8_t **rxbuff_readpos**
- uint8_t **df_mode**

The documentation for this struct was generated from the following file:

- W25Qxx.h

4.2 lfs Struct Reference

Collaboration diagram for lfs:



Data Structures

- struct lfs_free
- struct lfs_mlist

Data Fields

- lfs_cache_t **rcache**
- lfs_cache_t **pcache**
- lfs_block_t **root** [2]
- struct lfs::lfs_mlist * **mlist**
- uint32_t **seed**
- lfs_gstate_t **gstate**
- lfs_gstate_t **gdisk**
- lfs_gstate_t **gdelta**
- struct lfs::lfs_free **free**
- const struct lfs_config * **cfg**
- lfs_size_t **name_max**
- lfs_size_t **file_max**
- lfs_size_t **attr_max**

The documentation for this struct was generated from the following file:

- lfs.h

4.3 lfs_attr Struct Reference

Data Fields

- uint8_t **type**
- void * **buffer**
- lfs_size_t **size**

The documentation for this struct was generated from the following file:

- lfs.h

4.4 lfs_cache Struct Reference

```
#include <lfs.h>
```

Data Fields

- lfs_block_t **block**
- lfs_off_t **off**
- lfs_size_t **size**
- uint8_t * **buffer**

The documentation for this struct was generated from the following file:

- lfs.h

4.5 lfs_commit Struct Reference

Data Fields

- lfs_block_t **block**
- lfs_off_t **off**
- lfs_tag_t **ptag**
- uint32_t **crc**
- lfs_off_t **begin**
- lfs_off_t **end**

The documentation for this struct was generated from the following file:

- lfs.c

4.6 lfs_config Struct Reference

Data Fields

- void * **context**
- int(* **read**)(const struct lfs_config *c, lfs_block_t block, lfs_off_t off, void *buffer, lfs_size_t size)
- int(* **prog**)(const struct lfs_config *c, lfs_block_t block, lfs_off_t off, const void *buffer, lfs_size_t size)
- int(* **erase**)(const struct lfs_config *c, lfs_block_t block)
- int(* **sync**)(const struct lfs_config *c)
- lfs_size_t **read_size**
- lfs_size_t **prog_size**
- lfs_size_t **block_size**
- lfs_size_t **block_count**
- int32_t **block_cycles**
- lfs_size_t **cache_size**
- lfs_size_t **lookahead_size**
- void * **read_buffer**
- void * **prog_buffer**
- void * **lookahead_buffer**
- lfs_size_t **name_max**
- lfs_size_t **file_max**
- lfs_size_t **attr_max**
- lfs_size_t **metadata_max**

The documentation for this struct was generated from the following file:

- lfs.h

4.7 lfs_file::lfs_ctz Struct Reference

Data Fields

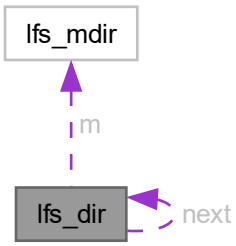
- lfs_block_t **head**
- lfs_size_t **size**

The documentation for this struct was generated from the following file:

- lfs.h

4.8 lfs_dir Struct Reference

Collaboration diagram for lfs_dir:



Data Fields

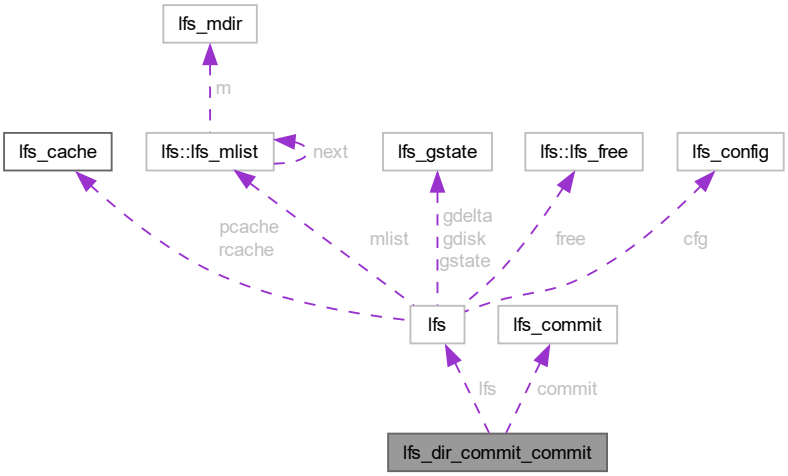
- struct lfs_dir * **next**
- uint16_t **id**
- uint8_t **type**
- lfs_mdir_t **m**
- lfs_off_t **pos**
- lfs_block_t **head** [2]

The documentation for this struct was generated from the following file:

- lfs.h

4.9 lfs_dir_commit_commit Struct Reference

Collaboration diagram for lfs_dir_commit_commit:



Data Fields

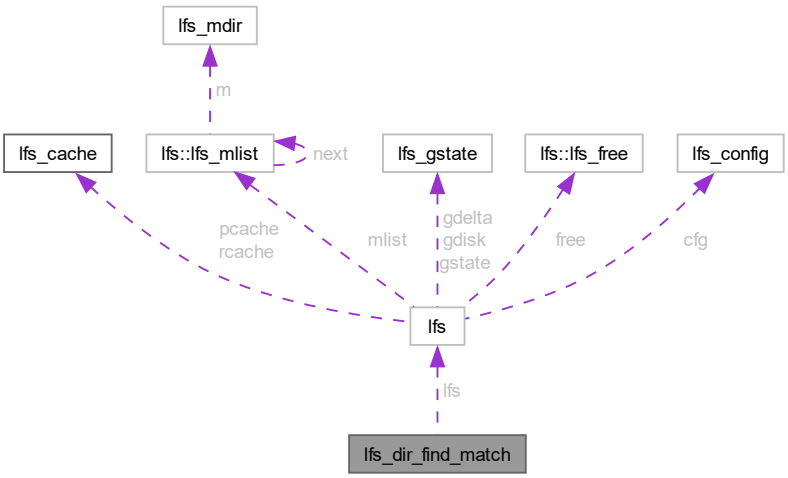
- lfs_t * **lfs**
- struct lfs_commit * **commit**

The documentation for this struct was generated from the following file:

- lfs.c

4.10 lfs_dir_find_match Struct Reference

Collaboration diagram for lfs_dir_find_match:



Data Fields

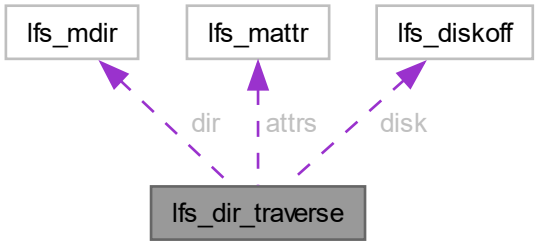
- lfs_t * **lfs**
- const void * **name**
- lfs_size_t **size**

The documentation for this struct was generated from the following file:

- lfs.c

4.11 lfs_dir_traverse Struct Reference

Collaboration diagram for lfs_dir_traverse:



Data Fields

- const lfs_mdir_t * **dir**
- lfs_off_t **off**
- lfs_tag_t **ptag**
- const struct lfs_mattr * **attrs**
- int **attrcount**
- lfs_tag_t **tmask**
- lfs_tag_t **ttag**
- uint16_t **begin**
- uint16_t **end**
- int16_t **diff**
- int(* **cb**)(void *data, lfs_tag_t tag, const void *buffer)
- void * **data**
- lfs_tag_t **tag**
- const void * **buffer**
- struct lfs_diskoff **disk**

The documentation for this struct was generated from the following file:

- lfs.c

4.12 lfs_diskoff Struct Reference

Data Fields

- lfs_block_t **block**
- lfs_off_t **off**

The documentation for this struct was generated from the following file:

- lfs.c

4.13 lfs_fcrc Struct Reference

Data Fields

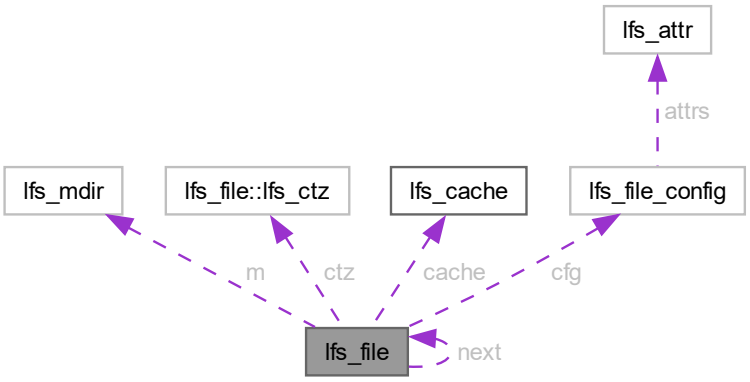
- lfs_size_t **size**
- uint32_t **crc**

The documentation for this struct was generated from the following file:

- lfs.c

4.14 lfs_file Struct Reference

Collaboration diagram for lfs_file:



Data Structures

- struct lfs_ctz

Data Fields

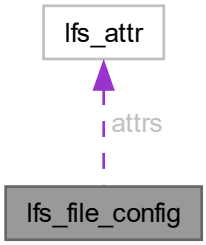
- struct lfs_file * **next**
- uint16_t **id**
- uint8_t **type**
- lfs_mdir_t **m**
- struct lfs_file::lfs_ctz **ctz**
- uint32_t **flags**
- lfs_off_t **pos**
- lfs_block_t **block**
- lfs_off_t **off**
- lfs_cache_t **cache**
- const struct lfs_file_config * **cfg**

The documentation for this struct was generated from the following file:

- lfs.h

4.15 lfs_file_config Struct Reference

Collaboration diagram for lfs_file_config:



Data Fields

- void * **buffer**
- struct lfs_attr * **attrs**
- lfs_size_t **attr_count**

The documentation for this struct was generated from the following file:

- lfs.h

4.16 lfs::lfs_free Struct Reference

Data Fields

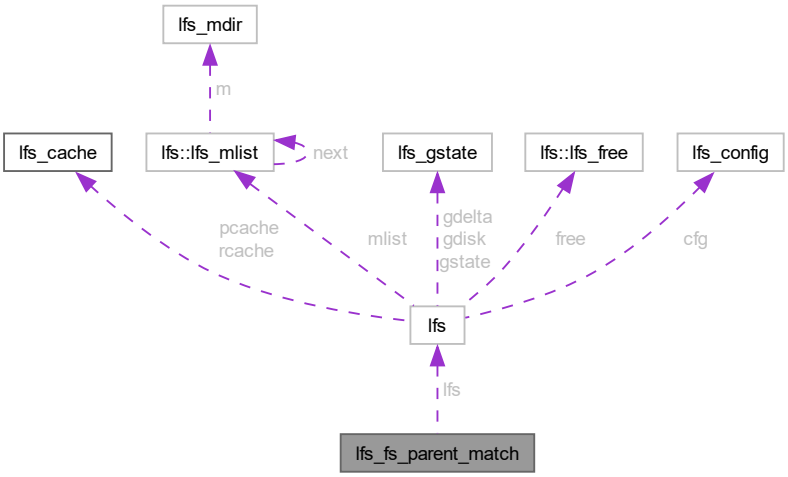
- lfs_block_t **off**
- lfs_block_t **size**
- lfs_block_t **i**
- lfs_block_t **ack**
- uint32_t * **buffer**

The documentation for this struct was generated from the following file:

- lfs.h

4.17 lfs_fs_parent_match Struct Reference

Collaboration diagram for lfs_fs_parent_match:



Data Fields

- lfs_t * **lfs**
- const lfs_block_t **pair** [2]

The documentation for this struct was generated from the following file:

- lfs.c

4.18 lfs_fsinfo Struct Reference

Data Fields

- uint32_t **disk_version**
- lfs_size_t **name_max**
- lfs_size_t **file_max**
- lfs_size_t **attr_max**

The documentation for this struct was generated from the following file:

- lfs.h

4.19 lfs_gstate Struct Reference

Data Fields

- uint32_t **tag**
- lfs_block_t **pair** [2]

The documentation for this struct was generated from the following file:

- lfs.h

4.20 lfs_info Struct Reference

Data Fields

- uint8_t **type**
- lfs_size_t **size**
- char **name** [LFS_NAME_MAX+1]

The documentation for this struct was generated from the following file:

- lfs.h

4.21 lfs_mattr Struct Reference

Data Fields

- lfs_tag_t **tag**
- const void * **buffer**

The documentation for this struct was generated from the following file:

- lfs.c

4.22 lfs_mdir Struct Reference

Data Fields

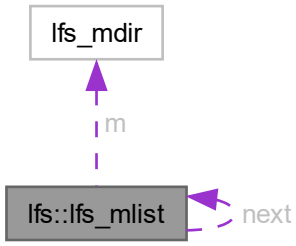
- lfs_block_t **pair** [2]
- uint32_t **rev**
- lfs_off_t **off**
- uint32_t **etag**
- uint16_t **count**
- bool **erased**
- bool **split**
- lfs_block_t **tail** [2]

The documentation for this struct was generated from the following file:

- lfs.h

4.23 lfs::lfs_mlist Struct Reference

Collaboration diagram for lfs::lfs_mlist:



Data Fields

- struct lfs_mlist * **next**
- uint16_t **id**
- uint8_t **type**
- lfs_mdir_t **m**

The documentation for this struct was generated from the following file:

- lfs.h

4.24 lfs_superblock Struct Reference

Data Fields

- uint32_t **version**
- lfs_size_t **block_size**
- lfs_size_t **block_count**
- lfs_size_t **name_max**
- lfs_size_t **file_max**
- lfs_size_t **attr_max**

The documentation for this struct was generated from the following file:

- lfs.h

4.25 sConsoleCommand_t Struct Reference

```
#include <Console.h>
```

Data Fields

- const char * **CommandName**
- char **CommandStr** [BUFFER_SIZE_8]
- volatile bool **IsCommandRaised**
- volatile bool **IsCommandServiced**
- pfConsoleCommandActor_t **pfConsoleCommandActor**

4.25.1 Detailed Description

The documentation for this struct was generated from the following file:

- Console.h

4.26 sElevatedPromptData_t Struct Reference

`#include <Console.h>`

Data Fields

- char **buf** [CONSOLE_PROMPT_BUFFER_SIZE]

4.26.1 Detailed Description

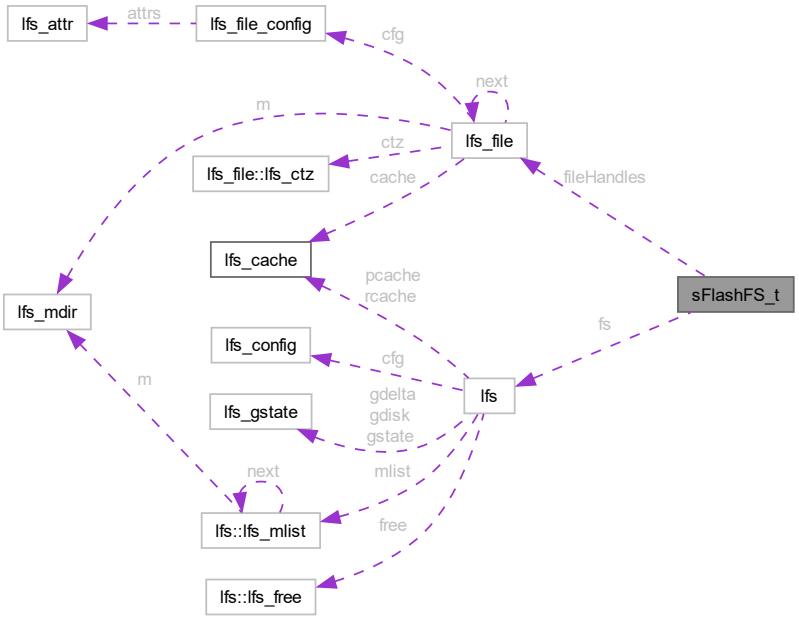
The documentation for this struct was generated from the following file:

- Console.h

4.27 sFlashFS_t Struct Reference

`#include <AppFlash_API.h>`

Collaboration diagram for sFlashFS_t:



Data Fields

- bool **IsMounted**
- lfs_t **fs**
- lfs_file_t **fileHandles** [eFS_MAX]

4.27.1 Detailed Description

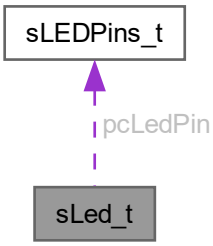
The documentation for this struct was generated from the following file:

- AppFlash_API.h

4.28 sLed_t Struct Reference

```
#include <TriColorLED.h>
```

Collaboration diagram for sLed_t:



Data Fields

- const sLEDPins_t * **pcLedPin**
- eLEDState_t **ledState**

4.28.1 Detailed Description

The documentation for this struct was generated from the following file:

- TriColorLED.h

4.29 sLEDPins_t Struct Reference

`#include <TriColorLED.h>`

Data Fields

- GPIO_TypeDef * **LEDPort**
- uint16_t **LEDPin**

4.29.1 Detailed Description

The documentation for this struct was generated from the following file:

- TriColorLED.h

4.30 sSDFS_t Struct Reference

`#include <AppSD_API.h>`

Data Fields

- bool **IsMounted**
- FATFS **fs**
- FIL **fileHandles** [eFS_MAX]

4.30.1 Detailed Description

The documentation for this struct was generated from the following file:

- AppSD_API.h

4.31 sSoftTimer_t Struct Reference

`#include <SoftTimer.h>`

Data Fields

- bool **IsArmed**
- bool **IsPeriodic**
- uint32_t **currentTicks**
- uint32_t **setTicks**
- pfCallBack_t **pfCallBack**

4.31.1 Detailed Description

The documentation for this struct was generated from the following file:

- SoftTimer.h

4.32 sTriColorIndication_t Struct Reference

`#include <TriColorLED.h>`

Data Fields

- bool **IsBlinkNeeded**
- eTriColorLEDStates_t **ledState**
- eTriColorLEDBlinkPeriod_t **ledBlinkPeriod**

4.32.1 Detailed Description

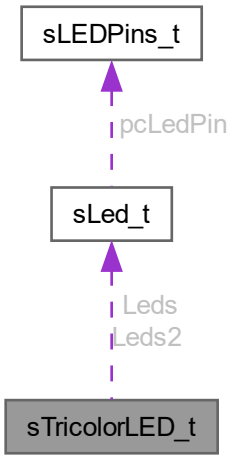
The documentation for this struct was generated from the following file:

- TriColorLED.h

4.33 sTricolorLED_t Struct Reference

`#include <TriColorLED.h>`

Collaboration diagram for sTricolorLED_t:



Data Fields

- bool **IsInitialized**
- bool **IsBlinkEnabled**
- size_t **blinkTicksSet**
- size_t **blinkTicksCurrent**
- eTriColorLEDStates_t **triColorLEDState**
- eTriColorLEDStates_t **prevOnLEDState**
- sLed_t **Leds** [eLED_MAX]
- sLed_t Leds2 [eLED_MAX]

4.33.1 Detailed Description

4.33.2 Field Documentation

4.33.2.1 Leds2

`sLed_t sTricolorLED_t::Leds2[eLED_MAX]`

Duplicate set of LED pins that shows the same indication

The documentation for this struct was generated from the following file:

- TriColorLED.h

4.34 sTriColorLEDState_t Struct Reference

`#include <TriColorLED.h>`

Data Fields

- eLEDState_t **LedState** [eLED_MAX]

4.34.1 Detailed Description

The documentation for this struct was generated from the following file:

- TriColorLED.h

4.35 w25qxx_t Struct Reference

Data Fields

- W25QXX_ID_t **ID**
- uint8_t **UniqID** [8]
- uint16_t **PageSize**
- uint32_t **PageCount**
- uint32_t **SectorSize**
- uint32_t **SectorCount**
- uint32_t **BlockSize**
- uint32_t **BlockCount**
- uint32_t **CapacityInKiloByte**
- uint8_t **StatusRegister1**
- uint8_t **StatusRegister2**
- uint8_t **StatusRegister3**
- uint8_t **Lock**

The documentation for this struct was generated from the following file:

- W25Qxx.h

Chapter 5

File Documentation

5.1 AppCommon.c File Reference

```
#include <stdint.h>
#include ¨stm32f1xx_hal.h¨
#include ¨AppCommon.h¨
#include ¨Version.h¨
#include ¨AppIndication.h¨
#include ¨Console.h¨
#include ¨AppConfiguration.h¨
```

Functions

- __attribute__ ((unused))
- static void AppCommon_PreResetRoutine ()
- eDeviceErrorCode_t AppCommon_GetErrorCode ()
- void AppCommon_AccumlateErrorCode (eDeviceErrorCode_t err)
- void AppCommon_ResetErrorCode ()
- void AppCommon_DetermineResetCause ()
- eDeviceResetCause_t AppCommon_GetCachedResetCause ()
- void AppCommon_HALErrorHandler ()
- void AppCommon_STMFaultHandler ()
- void __assert_func (const char *file, int line, const char *func, const char *failedexpr)
- void AppCommon_PrintDelimiter ()
- void AppCommon_PrintLineBreak ()
- void AppCommon_StartUpMessagePrint ()
- const char *const AppCommon_GetStatusString (int status)

Variables

- static eDeviceResetCause_t gDeviceResetCause = eRESET_CAUSE_UNKNOWN
- static eDeviceErrorCode_t gErrorCode = eERR_NO_ERRORS
- static const char * gcStatusStringHelper [eSTATUS_STRING_MAX]

5.1.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-06-03

Copyright

5.1.2 Function Documentation

5.1.2.1 __attribute__()

```
__attribute__ (
              (unused)  )
```

convert eDeviceResetCause_t enum to strings for printing

Delete File.

Write to a file in FatFS.

Parameters

pMe	FatFS wrapper instance
fileEnum	File to be written into
IsAppend	Pass true to write to file in append mode, false to write at the beginning
pInWriteBuf	data to be written is passed here
bufSize	size of buffer passed for writing

Returns

eStorageFSStatus_t

Parameters

pMe	FatFS wrapper instance
fileEnum	File to be deleted

Returns

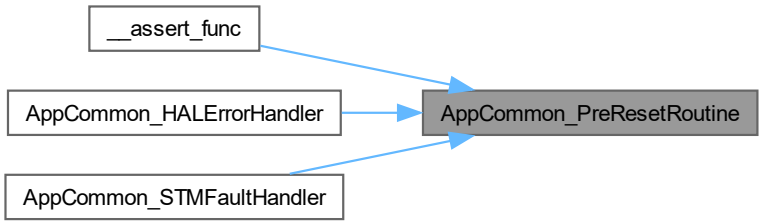
eStorageFSStatus_t

5.1.2.2 AppCommon_PreResetRoutine()

```
static void AppCommon_PreResetRoutine ( )  [static]
```

Function to be invoked prior to resetting the system to recover from an hard fault/error.

< Device has been reset so set up-time back to ZeroHere is the caller graph for this function:



5.1.2.3 AppCommon_GetErrorCode()

```
eDeviceErrorCode_t AppCommon_GetErrorCode ( )
```

Setter for gErrorCode, previous values are unaffected.

Returns

eDeviceErrorCode_t

Here is the caller graph for this function:

5.1.2.4 AppCommon_AccumlateErrorCode()

```
void AppCommon_AccumlateErrorCode (
            eDeviceErrorCode_t err )
```

Setter for gErrorCode, previous values are unaffected.

Here is the caller graph for this function:

5.1.2.5 AppCommon_ResetErrorCode()

```
void AppCommon_ResetErrorCode ( )
```

Reset error Code.

Here is the caller graph for this function:

5.1.2.6 AppCommon_DetermineResetCause()

```
void AppCommon_DetermineResetCause ( )
```

Function to determine why the device was reset, also sets global gDeviceResetCause.

Note

should be called before peripheral initialization

5.1.2.7 AppCommon_GetCachedResetCause()

```
eDeviceResetCause_t AppCommon_GetCachedResetCause ( )
```

Getter for gDeviceResetCause.

Note

Assumes AppCommon_DetermineResetCause is called at startup

Returns

eDeviceResetCause_t

5.1.2.8 AppCommon_HALErrorHandler()

```
void AppCommon_HALErrorHandler ( )
```

STM HAL error handler.

Warning

This function resets the device

Here is the call graph for this function:

5.1.2.9 AppCommon_STMFaultHandler()

```
void AppCommon_STMFaultHandler ( )
```

STM Fault error handler.

Warning

This function resets the device

Here is the call graph for this function:

5.1.2.10 __assert_func()

```
void __assert_func (
            const char * file,
            int line,
            const char * func,
            const char * failedexpr )
```

Assert failure error handler.

Warning

This function resets the device

Here is the call graph for this function:

5.1.2.11 AppCommon_PrintDelimiter()

```
void AppCommon_PrintDelimiter ( )
```

Utility function to print delimiter used in the console.

Here is the call graph for this function:

Here is the caller graph for this function:

5.1.2.12 AppCommon_PrintLineBreak()

```
void AppCommon_PrintLineBreak ( )
```

Utility function to print delimiter used in the console.

Here is the call graph for this function:

Here is the caller graph for this function:

5.1.2.13 AppCommon_StartUpMessagePrint()

```
void AppCommon_StartUpMessagePrint ( )
```

Start up message print.

Here is the call graph for this function:

Here is the caller graph for this function:

5.1.2.14 AppCommon_GetStatusString()

```
const char *const AppCommon_GetStatusString (
            int status )
```

Utility function returns ¨Success¨ or ¨failure¨ based on status code passed.

See also

gcStatusStringHelper

Parameters

status	0 is success across all modules in the code-base , non 0 is treated as failure

Returns

const char∗ const

Here is the caller graph for this function:

5.1.3 Variable Documentation

5.1.3.1 gDeviceResetCause

```
eDeviceResetCause_t gDeviceResetCause = eRESET_CAUSE_UNKNOWN  [static]
```

Global that maintains reset cause

5.1.3.2 gErrorCode

```
eDeviceErrorCode_t gErrorCode = eERR_NO_ERRORS  [static]
```

Global that maintains error cause

5.1.3.3 gcStatusStringHelper

```
const char* gcStatusStringHelper[eSTATUS_STRING_MAX]  [static]
```

Initial value:

```
=
{
        [eSUCCESS_STRING] = "Success",
        [eFAILURE_STRING] = "Failure"
}
```

Map eStatusPrintHelperEnum_t to string. Used by AppCommon_GetStatusString.

5.2 AppCommon.h File Reference

Macros

- #define LOOP_FOREVER {while(1);}

Enumerations

- enum eStatusPrintHelperEnum_t { **eSUCCESS_STRING** , **eFAILURE_STRING** , **eSTATUS_STRING_MAX** }
- enum eDeviceResetCause_t {
  eRESET_CAUSE_UNKNOWN = 0 , **eRESET_CAUSE_WAKEUP** , **eRESET_CAUSE_LOW_POWER_RESET** , **eRESET_CAUSE_WINDOW_WATCHDOG_RESET** ,
  eRESET_CAUSE_INDEPENDENT_WATCHDOG_RESET , **eRESET_CAUSE_SOFTWARE_RESET** , **eRESET_CAUSE_POWER_ON_POWER_DOWN_RESET** , **eRESET_CAUSE_EXTERNAL_RESET_PIN_RESET** ,
  eRESET_CAUSE_BROWNOUT_RESET , **eRESET_CAUSE_MAX** }
- enum eDeviceErrorCode_t {
  eERR_NO_ERRORS = 0x0000 , **eERR_SDCARD_NOT_FOUND** = 0x0001 , **eERR_SDCARD_FILE_NOT_FOUND** = 0x0002 , **eERR_FLASH_NOT_FOUND** = 0x0004 ,
  eERR_FLASH_TRANSFER_FAILURE = 0x0008 , **eERR_CRC_FAILURE** = 0x0010 , **eERR_UNUSED_1** = 0x0020 , **eERR_UNUSED_2** = 0x0040 ,
  eERR_UNUSED_3 = 0x0080 , **eERR_UNUSED_4** = 0x0100 , **eERR_UNUSED_5** = 0X0200 , **eERR_UNUSED_6** = 0x0400 ,
  eERR_UNUSED_7 = 0x0800 , **eERR_STM_HAL_FAILURE** = 0x1000 , **eERR_ARM_FAULT** = 0x2000 , **eERR_ASSERTION_FAILURE** = 0x4000 ,
  eERR_SLEEP_FAILURE = 0x8000 }

Functions

- void AppCommon_PrintDelimiter ()
- void AppCommon_PrintLineBreak ()
- void AppCommon_StartUpMessagePrint ()
- void AppCommon_HALErrorHandler ()
- void AppCommon_STMFaultHandler ()
- const char *const AppCommon_GetStatusString (int status)
- void __assert_func (const char *file, int line, const char *func, const char *failedexpr)
- void AppCommon_DetermineResetCause ()
- eDeviceResetCause_t AppCommon_GetCachedResetCause ()
- eDeviceErrorCode_t AppCommon_GetErrorCode ()
- void AppCommon_AccumlateErrorCode (eDeviceErrorCode_t err)
- void AppCommon_ResetErrorCode ()

5.2.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-06-03

Copyright

5.2.2 Macro Definition Documentation

5.2.2.1 LOOP_FOREVER

```
#define LOOP_FOREVER {while(1);}
```

Utility MACRO to loop forever

5.2.3 Enumeration Type Documentation

5.2.3.1 eStatusPrintHelperEnum_t

```
enum eStatusPrintHelperEnum_t
```

Enum for status helper.

5.2.3.2 eDeviceResetCause_t

```
enum eDeviceResetCause_t
```

Possible STM32 system reset causes.

5.2.3.3 eDeviceErrorCode_t

```
enum eDeviceErrorCode_t
```

Possible Errors in application.

5.2.4 Function Documentation

5.2.4.1 AppCommon_PrintDelimiter()

```
void AppCommon_PrintDelimiter ( )
```

Utility function to print delimiter used in the console.

Here is the call graph for this function:

Here is the caller graph for this function:

5.2.4.2 AppCommon_PrintLineBreak()

```
void AppCommon_PrintLineBreak ( )
```

Utility function to print delimiter used in the console.

Here is the call graph for this function:

Here is the caller graph for this function:

5.2.4.3 AppCommon_StartUpMessagePrint()

```
void AppCommon_StartUpMessagePrint ( )
```

Start up message print.

Here is the call graph for this function:

Here is the caller graph for this function:

5.2.4.4 AppCommon_HALErrorHandler()

```
void AppCommon_HALErrorHandler ( )
```

STM HAL error handler.

Warning

This function resets the device

Here is the call graph for this function:

5.2.4.5 AppCommon_STMFaultHandler()

```
void AppCommon_STMFaultHandler ( )
```

STM Fault error handler.

Warning

This function resets the device

Here is the call graph for this function:

5.2.4.6 AppCommon_GetStatusString()

```
const char *const AppCommon_GetStatusString (
            int status )
```

Utility function returns ¨Success¨ or ¨failure¨ based on status code passed.

See also

gcStatusStringHelper

Parameters

status	0 is success across all modules in the code-base , non 0 is treated as failure

Returns

const char* const

Here is the caller graph for this function:

5.2.4.7 __assert_func()

```
void __assert_func (
            const char * file,
            int line,
            const char * func,
            const char * failedexpr )
```

Assert failure error handler.

Warning

This function resets the device

Here is the call graph for this function:

5.2.4.8 AppCommon_DetermineResetCause()

```
void AppCommon_DetermineResetCause ( )
```

Function to determine why the device was reset, also sets global gDeviceResetCause.

Note

should be called before peripheral initialization

5.2.4.9 AppCommon_GetCachedResetCause()

```
eDeviceResetCause_t AppCommon_GetCachedResetCause ( )
```

Getter for gDeviceResetCause.

Note

Assumes AppCommon_DetermineResetCause is called at startup

Returns

eDeviceResetCause_t

5.2.4.10 AppCommon_GetErrorCode()

```
eDeviceErrorCode_t AppCommon_GetErrorCode ( )
```

Setter for gErrorCode, previous values are unaffected.

Returns

eDeviceErrorCode_t

Here is the caller graph for this function:

5.2.4.11 AppCommon_AccumlateErrorCode()

```
void AppCommon_AccumlateErrorCode (
            eDeviceErrorCode_t err )
```

Setter for gErrorCode, previous values are unaffected.

Here is the caller graph for this function:

5.2.4.12 AppCommon_ResetErrorCode()

```
void AppCommon_ResetErrorCode ( )
```

Reset error Code.

Here is the caller graph for this function:

5.3 AppCommon.h

Go to the documentation of this file.

```
00001
00013
00014 #ifndef APPCOMMON_APPCOMMON_H_
00015 #define APPCOMMON_APPCOMMON_H_
00016
00018
00019 #define LOOP_FOREVER {while(1);}
00022
00027 typedef enum
00028 {
00029     eSUCCESS_STRING,
00030     eFAILURE_STRING,
00031     eSTATUS_STRING_MAX
00032 }eStatusPrintHelperEnum_t;
00033
00038 typedef enum
00039 {
00040     eRESET_CAUSE_UNKNOWN = 0,
```

```
00041     eRESET_CAUSE_WAKEUP,
00042     eRESET_CAUSE_LOW_POWER_RESET,
00043     eRESET_CAUSE_WINDOW_WATCHDOG_RESET,
00044     eRESET_CAUSE_INDEPENDENT_WATCHDOG_RESET,
00045     eRESET_CAUSE_SOFTWARE_RESET,
00046     eRESET_CAUSE_POWER_ON_POWER_DOWN_RESET,
00047     eRESET_CAUSE_EXTERNAL_RESET_PIN_RESET,
00048     eRESET_CAUSE_BROWNOUT_RESET,
00049     eRESET_CAUSE_MAX
00050 } eDeviceResetCause_t;
00051
00052
00057 typedef enum
00058 {
00059     eERR_NO_ERRORS                  = 0x0000,
00060
00061     eERR_SDCARD_NOT_FOUND           = 0x0001,
00062     eERR_SDCARD_FILE_NOT_FOUND      = 0x0002,
00063
00064     eERR_FLASH_NOT_FOUND            = 0x0004,
00065     eERR_FLASH_TRANSFER_FAILURE     = 0x0008,
00066
00067     eERR_CRC_FAILURE                = 0x0010,
00068
00069     eERR_UNUSED_1                   = 0x0020,
00070     eERR_UNUSED_2                   = 0x0040,
00071     eERR_UNUSED_3                   = 0x0080,
00072     eERR_UNUSED_4                   = 0x0100,
00073     eERR_UNUSED_5                   = 0X0200,
00074     eERR_UNUSED_6                   = 0x0400,
00075     eERR_UNUSED_7                   = 0x0800,
00076
00077     eERR_STM_HAL_FAILURE            = 0x1000,
00078     eERR_ARM_FAULT                  = 0x2000,
00079     eERR_ASSERTION_FAILURE          = 0x4000,
00080     eERR_SLEEP_FAILURE              = 0x8000,
00081
00082 }eDeviceErrorCode_t;
00083
00084
00086
00087
00088 void AppCommon_PrintDelimiter();
00089 void AppCommon_PrintLineBreak();
00090 void AppCommon_StartUpMessagePrint();
00091 void AppCommon_HALErrorHandler();
00092 void AppCommon_STMFaultHandler();
00093 const char* const AppCommon_GetStatusString(int status);
00094 void __assert_func(const char *file, int line, const char *func, const char *failedexpr);
00095
00096 void AppCommon_DetermineResetCause();
00097 eDeviceResetCause_t AppCommon_GetCachedResetCause();
00098
00099 eDeviceErrorCode_t AppCommon_GetErrorCode();
00100 void AppCommon_AccumlateErrorCode(eDeviceErrorCode_t err);
00101 void AppCommon_ResetErrorCode();
00102
00104
00105 #endif /* APPCOMMON_APPCOMMON_H_ */
```

5.4 AppConfiguration.c File Reference

`#include ¨AppConfiguration.h¨`

5.4.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-05-24

Copyright

5.5 AppConfiguration.h File Reference

5.5.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-05-24

Copyright

5.6 AppConfiguration.h

Go to the documentation of this file.

```
00001
00013
00014 #ifndef APPCONFIGURATION_APPCONFIGURATION_H_
00015 #define APPCONFIGURATION_APPCONFIGURATION_H_
00016
00018
00021 //#define ENABLE_DEBUG_PRINT               /**< Enabling this macro will redirect @ref DEBUG_PRINT to
      @ref Console_Print */
00022 //#define ENABLE_TESTS_DEFINITIONS         /**< If this is enabled then tests defined for individual
      modules are defined*/
00023 //#define FORCE_DISABLE_FILE_CRC_CHECK     /**< CRC of the SD card an Flash file copy will be
      computed and compared by default, define this variable to skip CRC check*/
00024
00025
00027
00028
00029 #endif /* APPCONFIGURATION_APPCONFIGURATION_H_ */
```

5.7 Version.h File Reference

```
#include <stdio.h>
```

Macros

- #define **FW_VERSION_MAJOR** (1)
- #define **FW_VERSION_MINOR** (0)
- #define **HARDWARE_VERSION_MAJOR** (1)
- #define **HARDWARE_VERSION_MINOR** (1)
- #define **COMPILE_DATE** __DATE__
- #define **COMPILE_TIME** __TIME__

5.7.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-05-24

Copyright

Copyright (c) 2023

5.8 Version.h

Go to the documentation of this file.

```
00001
00013
00014 #ifndef APPCONFIGURATION_VERSION_H_
00015 #define APPCONFIGURATION_VERSION_H_
00016
00018
00019 #include <stdio.h>
00020
00022
00023 #define FW_VERSION_MAJOR        (1)
00024 #define FW_VERSION_MINOR        (0)
00025
00026 #define HARDWARE_VERSION_MAJOR  (1)
00027 #define HARDWARE_VERSION_MINOR  (1)
00028
00029 #define COMPILE_DATE            __DATE__
00030 #define COMPILE_TIME            __TIME__
00031
00033
00034 #endif /* APPCONFIGURATION_VERSION_H_ */
```

5.9 AppIndication.c File Reference

```
#include <assert.h>
#include ¨AppIndication.h¨
#include ¨TriColorLED.h¨
```

Functions

- void AppIndicate_Init ()
- void AppIndicate_DeInit ()
- void AppIndicate_SetState (eAppIndicationStates_t state)
- void AppIndicate_RevertState ()

Variables

- static sTriColorIndication_t gcAppIndicationTable [eIND_MAX]
- static eAppIndicationStates_t gCurrentState = eIND_NONE
- static eAppIndicationStates_t gPrevState = eIND_NONE

5.9.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2024-05-09

Copyright

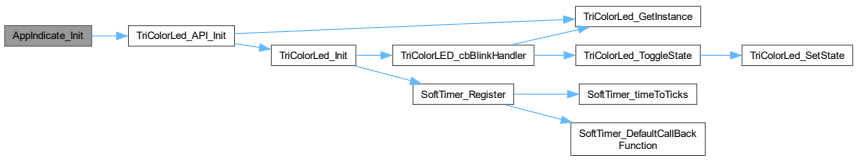
5.9.2 Function Documentation

5.9.2.1 AppIndicate_Init()

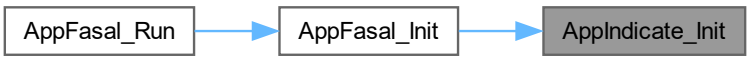
```
void AppIndicate_Init ( )
```

Wrapper around Indication medium initialization.

Here is the call graph for this function:



Here is the caller graph for this function:

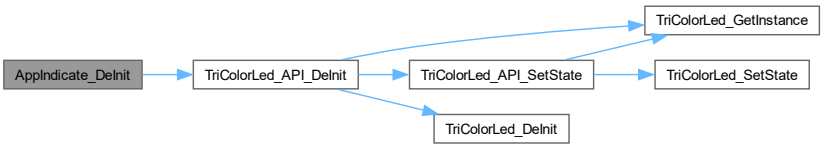


5.9.2.2 AppIndicate_DeInit()

```
void AppIndicate_DeInit ( )
```

Wrapper around Indication medium De-initialization.

Here is the call graph for this function:



5.9.2.3 AppIndicate_SetState()

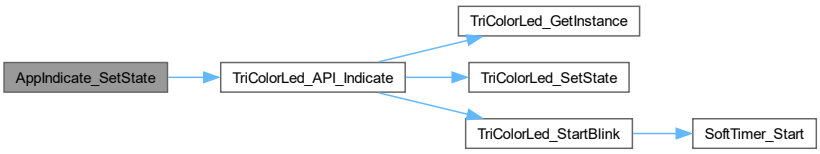
```
void AppIndicate_SetState (
            eAppIndicationStates_t state )
```

Set Indication medium state.

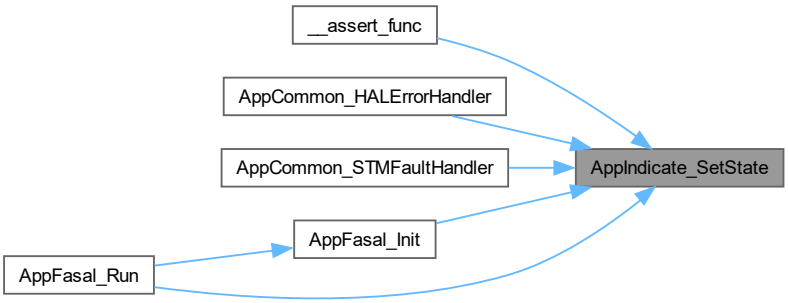
Parameters

state	Which application state to set to, gcAppIndicationTable

Here is the call graph for this function:



Here is the caller graph for this function:

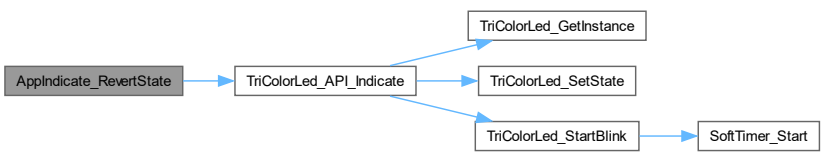


5.9.2.4 AppIndicate_RevertState()

```
void AppIndicate_RevertState ( )
```

Revert to previous indication state. For preserving previous Indication state.

Here is the call graph for this function:



5.9.3 Variable Documentation

5.9.3.1 gcAppIndicationTable

```
sTriColorIndication_t gcAppIndicationTable[eIND_MAX]  [static]
```

Configuration table that determine how application states correspond to TriColor LED states.

5.9.3.2 gCurrentState

```
eAppIndicationStates_t gCurrentState = eIND_NONE  [static]
```

Current Indication state maintained in this global

5.9.3.3 gPrevState

```
eAppIndicationStates_t gPrevState = eIND_NONE  [static]
```

Previous Indication state maintained here

5.10 AppIndication.h File Reference

Enumerations

- enum eAppIndicationStates_t {
  eIND_NO_CHANGE , **eIND_NONE** , **eIND_RED_0** , **eIND_GREEN_0** ,
  eIND_BLUE_0 , **eIND_YELLOW_0** , **eIND_MAGENTA_0** , **eIND_CYAN_0** ,
  eIND_WHITE_0 , **eIND_RED_250MS** , **eIND_GREEN_250MS** , **eIND_BLUE_250MS** ,
  eIND_YELLOW_250MS , **eIND_MAGENTA_250MS** , **eIND_CYAN_250MS** , **eIND_WHITE_250MS** ,
  eIND_RED_500MS , **eIND_GREEN_500MS** , **eIND_BLUE_500MS** , **eIND_YELLOW_500MS** ,
  eIND_MAGENTA_500MS , **eIND_CYAN_500MS** , **eIND_WHITE_500MS** , **eIND_RED_1000MS** ,
  eIND_GREEN_1000MS , **eIND_BLUE_1000MS** , **eIND_YELLOW_1000MS** , **eIND_MAGENTA_1000MS** ,
  eIND_CYAN_1000MS , **eIND_WHITE_1000MS** , **eIND_RED_2000MS** , **eIND_GREEN_2000MS** ,
  eIND_BLUE_2000MS , **eIND_YELLOW_2000MS** , **eIND_MAGENTA_2000MS** , **eIND_CYAN_2000MS** ,
  eIND_WHITE_2000MS , **eIND_MAX** }

Functions

- void AppIndicate_Init ()
- void AppIndicate_SetState (eAppIndicationStates_t state)
- void AppIndicate_RevertState ()

5.10.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2024-05-09

Copyright

Copyright (c) 2024

5.10.2 Enumeration Type Documentation

5.10.2.1 eAppIndicationStates_t

```
enum eAppIndicationStates_t
```

Application events that require LED indication are enumerated here gcAppIndicationTable.

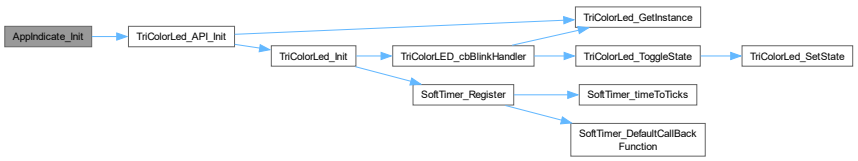
5.10.3 Function Documentation

5.10.3.1 AppIndicate_Init()

```
void AppIndicate_Init ( )
```

Wrapper around Indication medium initialization.

Here is the call graph for this function:



Here is the caller graph for this function:

5.10.3.2 AppIndicate_SetState()

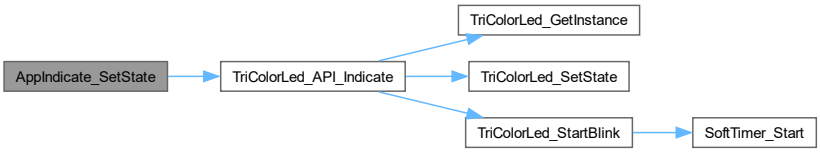
```
void AppIndicate_SetState (
            eAppIndicationStates_t state )
```

Set Indication medium state.

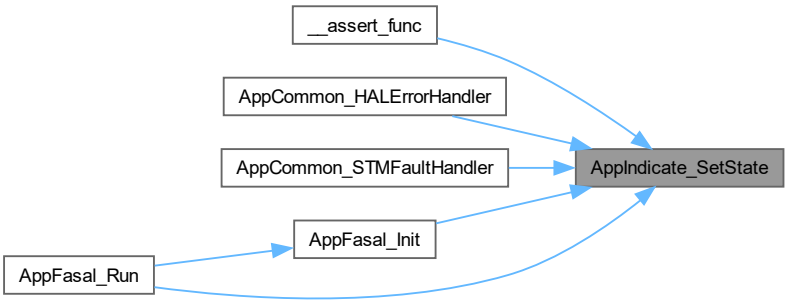
Parameters

state	Which application state to set to, gcAppIndicationTable

Here is the call graph for this function:



Here is the caller graph for this function:

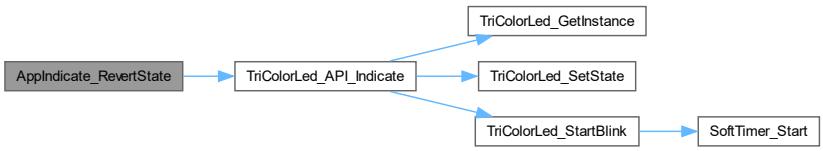


5.10.3.3 AppIndicate_RevertState()

```
void AppIndicate_RevertState ( )
```

Revert to previous indication state. For preserving previous Indication state.

Here is the call graph for this function:



5.11 AppIndication.h

Go to the documentation of this file.

```
00001
00013
00014 #ifndef APPCOMMON_APPINDICATION_H_
00015 #define APPCOMMON_APPINDICATION_H_
00016
00018
00023 typedef enum
00024 {
00025     eIND_NO_CHANGE,
00026     eIND_NONE,
00027
00028     eIND_RED_0,
00029     eIND_GREEN_0,
00030     eIND_BLUE_0,
00031     eIND_YELLOW_0,
00032     eIND_MAGENTA_0,
00033     eIND_CYAN_0,
00034     eIND_WHITE_0,
00035
00036     eIND_RED_250MS,
00037     eIND_GREEN_250MS,
00038     eIND_BLUE_250MS,
00039     eIND_YELLOW_250MS,
00040     eIND_MAGENTA_250MS,
00041     eIND_CYAN_250MS,
00042     eIND_WHITE_250MS,
00043
00044     eIND_RED_500MS,
00045     eIND_GREEN_500MS,
00046     eIND_BLUE_500MS,
00047     eIND_YELLOW_500MS,
00048     eIND_MAGENTA_500MS,
00049     eIND_CYAN_500MS,
00050     eIND_WHITE_500MS,
00051
00052     eIND_RED_1000MS,
00053     eIND_GREEN_1000MS,
00054     eIND_BLUE_1000MS,
00055     eIND_YELLOW_1000MS,
00056     eIND_MAGENTA_1000MS,
00057     eIND_CYAN_1000MS,
00058     eIND_WHITE_1000MS,
00059
00060     eIND_RED_2000MS,
00061     eIND_GREEN_2000MS,
00062     eIND_BLUE_2000MS,
00063     eIND_YELLOW_2000MS,
00064     eIND_MAGENTA_2000MS,
00065     eIND_CYAN_2000MS,
00066     eIND_WHITE_2000MS,
00067
00068     eIND_MAX,
00069 }eAppIndicationStates_t;
00070
00072
00073 void AppIndicate_Init();
00074 void AppIndicate_SetState(eAppIndicationStates_t state);
00075 void AppIndicate_RevertState();
00076
00078
00079 #endif /* APPCOMMON_APPINDICATION_H_ */
```

5.12 AppProfiler.c File Reference

```
#include ¨AppProfiler.h¨
#include ¨stm32f1xx_hal.h¨
```

Functions

- void **AppProfiler_StartExecutionTimeMeasurement** ()
- uint32_t AppProfiler_GetExecutionTimeMS ()

5.12.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2024-06-13

Copyright

Copyright (c) 2023

5.12.2 Function Documentation

5.12.2.1 AppProfiler_GetExecutionTimeMS()

```
uint32_t AppProfiler_GetExecutionTimeMS ( )
```

Get function execution time.

Note

This function expects AppProfiler_StartExecutionTimeMeasurement to be invoked prior to use

Returns

execution time in milliseconds

5.13 AppProfiler.h File Reference

```
#include <stdint.h>
```

Functions

- void **AppProfiler_StartExecutionTimeMeasurement** ()
- uint32_t AppProfiler_GetExecutionTimeMS ()

5.13.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2024-06-13

Copyright

5.13.2 Function Documentation

5.13.2.1 AppProfiler_GetExecutionTimeMS()

```
uint32_t AppProfiler_GetExecutionTimeMS ( )
```

Get function execution time.

Note

This function expects AppProfiler_StartExecutionTimeMeasurement to be invoked prior to use

Returns

execution time in milliseconds

5.14 AppProfiler.h

Go to the documentation of this file.

```
00001
00013
00014 #ifndef APPCOMMON_APPUTILITY_APPPROFILER_APPPROFILER_H_
00015 #define APPCOMMON_APPUTILITY_APPPROFILER_APPPROFILER_H_
00016
00018
00019 #include <stdint.h>
00020
00022
00023 void AppProfiler_StartExecutionTimeMeasurement();
00024 uint32_t AppProfiler_GetExecutionTimeMS();
00025
00027
00028 #endif /* APPCOMMON_APPUTILITY_APPPROFILER_APPPROFILER_H_ */
```

5.15 DebugPrint.h File Reference

```
#include ¨AppConfiguration.h¨
#include ¨Console.h¨
```

Macros

- #define **DEBUG_PRINT**(...)

5.15.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-05-24

Copyright

5.16 DebugPrint.h

Go to the documentation of this file.

```
00001
00012 #ifndef UTILITY_DEBUGPRINT_H_
00013 #define UTILITY_DEBUGPRINT_H_
00014
00016
00017 #include "AppConfiguration.h"
00018 #include "Console.h"
00019
00021
00022 #ifdef ENABLE_DEBUG_PRINT
00023 #define DEBUG_PRINT Console_Print
00024 #else
00025 #define DEBUG_PRINT(...)
00026 #endif
00027
00028
00029 #endif /* UTILITY_DEBUGPRINT_H_ */
```

5.17 SoftTimer.c File Reference

```
#include <assert.h>
#include ¨SoftTimer/SoftTimer.h¨
```

Functions

- void SoftTimer_cbPeriodicCheck ()
- static void SoftTimer_DefaultCallBackFunction ()
- static uint32_t SoftTimer_timeToTicks (uint32_t timeMs)
- void SoftTimer_Init ()
- __attribute__ ((unused))
- void SoftTimer_DeInit ()
- void SoftTimer_Register (eSoftTimerID_t id, uint32_t timeOut, bool IsPeriodic, pfCallBack_t callbackFunction)
- void SoftTimer_Start (eSoftTimerID_t id, bool IsStartNeeded)
- void SoftTimer_Pause (eSoftTimerID_t id, bool IsPauseNeeded)
- void SoftTimer_StartAll (bool IsStartNeeded)
- bool SofTimer_IsExpired (eSoftTimerID_t id)
- void SoftTimer_AperiodicTimerSet (uint32_t timeOut)
- bool SoftTimer_HasAperiodicTimerExpired ()
- void SoftTimer_DelayMS (uint32_t delayMS)

Variables

- static volatile sSoftTimer_t gvSoftTimers [eSOFT_TIMER_MAX]

5.17.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.2

Date

2023-05-28

Copyright

5.17.2 Function Documentation

5.17.2.1 SoftTimer_cbPeriodicCheck()

```
void SoftTimer_cbPeriodicCheck ( )
```

Periodic call from soft-timer that checks if any registered timers have expired and invokes the registered call back HAL_TIM_PeriodElapsedCallback.

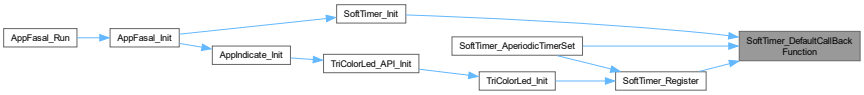
Here is the caller graph for this function:

5.17.2.2 SoftTimer_DefaultCallBackFunction()

```
static void SoftTimer_DefaultCallBackFunction ( )  [static]
```

Default call back for soft-timers without any executors.

Here is the caller graph for this function:



5.17.2.3 SoftTimer_timeToTicks()

```
static uint32_t SoftTimer_timeToTicks (
            uint32_t timeMs )  [static]
```

Utility function to convert timeinMS to soft-timer ticks count.

Parameters

timeMs	

Returns

uint32_t

Here is the caller graph for this function:

5.17.2.4 SoftTimer_Init()

```
void SoftTimer_Init ( )
```

Initialize soft-timer.

Here is the call graph for this function:

Here is the caller graph for this function:

5.17.2.5 __attribute__()

```
__attribute__ (
              (unused)  )
```

Get time left for expiry.

Parameters

id	ID of soft-timer

Returns

uint32_t time left for soft-timer expire in ms

5.17.2.6 SoftTimer_DeInit()

```
void SoftTimer_DeInit ( )
```

DeInitialize soft-timer.

5.17.2.7 SoftTimer_Register()

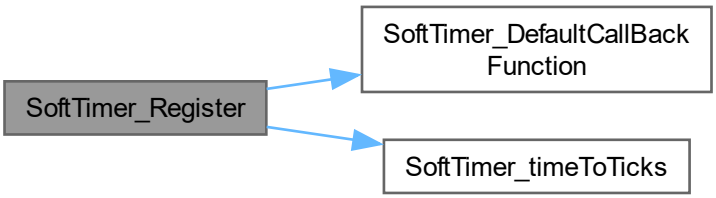
```
void SoftTimer_Register (
            eSoftTimerID_t id,
            uint32_t timeOut,
            bool IsPeriodic,
            pfCallBack_t callbackFunction )
```

Register a periodic function with soft-timer, create ID under eSoftTimerID and attach call-back and time period through this function.

Parameters

id	
timeOut	
IsPeriodic	
callbackFunction	

Here is the call graph for this function:



Here is the caller graph for this function:

5.17.2.8 SoftTimer_Start()

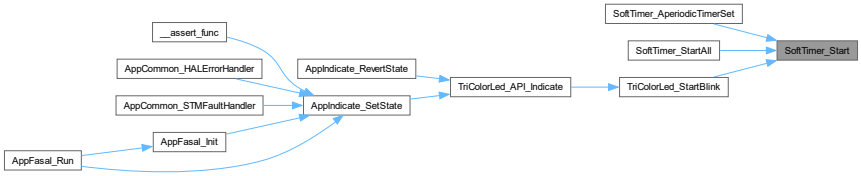
```
void SoftTimer_Start (
            eSoftTimerID_t id,
            bool IsStartNeeded )
```

Start soft-timer.

Parameters

id	
IsStartNeeded	

Here is the caller graph for this function:



5.17.2.9 SoftTimer_Pause()

```
void SoftTimer_Pause (
            eSoftTimerID_t id,
            bool IsPauseNeeded )
```

Function to disable and enable soft-timer for creating critical sections.

Parameters

id	ID of the soft-timer to pause/resume
IsPauseNeeded	Pause if true, resume if false

5.17.2.10 SoftTimer_StartAll()

```
void SoftTimer_StartAll (
            bool IsStartNeeded )
```

Collectively start all registered soft-timers.

Parameters

IsStartNeeded	

Here is the call graph for this function:

5.17.2.11 SofTimer_IsExpired()

```
bool SofTimer_IsExpired (
            eSoftTimerID_t id )
```

Check if soft-timers has expired.

Parameters

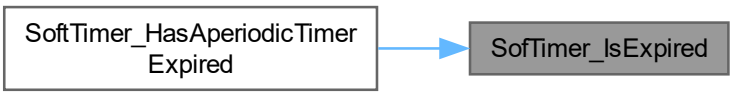
id	id of soft-timer to be checked

Returns

true if soft-timer has expired

false if V has not expired / has not been enabled

Here is the caller graph for this function:



5.17.2.12 SoftTimer_AperiodicTimerSet()

```
void SoftTimer_AperiodicTimerSet (
            uint32_t timeOut )
```

Set Aperiodic timer instance useful for setting timeouts.

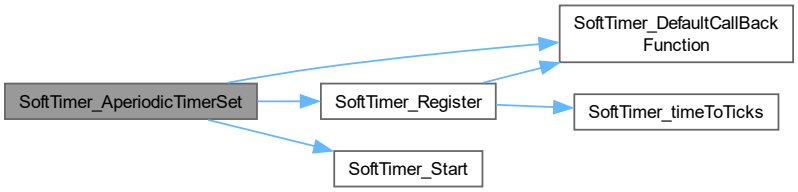
Note

eGENERIC_COUNT_DOWN_TIMER must not be used for other periodic timers

Parameters

timeOut	time out to set the aperiodic timer

Here is the call graph for this function:



5.17.2.13 SoftTimer_HasAperiodicTimerExpired()

```
bool SoftTimer_HasAperiodicTimerExpired ( )
```

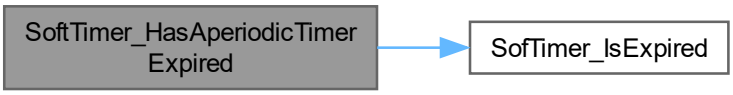
Check if aperiodic timer has expired.

Returns

true if timer has expired

false if timer has not expired/ Registered / started

Here is the call graph for this function:



5.17.2.14 SoftTimer_DelayMS()

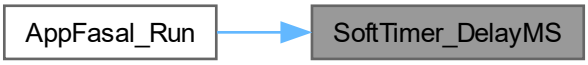
```
void SoftTimer_DelayMS (
            uint32_t delayMS )
```

Wrapper around HAL_Delay delay.

Parameters

delayMS	delay needed in mS

Here is the caller graph for this function:



5.17.3 Variable Documentation

5.17.3.1 gvSoftTimers

```
volatile sSoftTimer_t gvSoftTimers[eSOFT_TIMER_MAX]  [static]
```

All soft-timer instances are captured here.

5.18 SoftTimer.h File Reference

```
#include <stdint.h>
#include <stdbool.h>
#include "tim.h"
```

Data Structures

- struct sSoftTimer_t

Macros

- #define **SOFTTIMER_TIMER_INSTANCE** (&htim6)
- #define **SOFTTIMER_IRQ** (TIM6_IRQn)
- #define **SOFTTIMER_OVERFLOW_PERIOD_MS** (10)

Typedefs

- typedef void(* **pfCallBack_t**) (void)

Enumerations

- enum eSoftTimerID_t { eGENERIC_COUNT_DOWN_TIMER , **ePUSH_BUTTON_TIMER** , **eDEBUG_LED_SOFT_TIMER** , **eSOFT_TIMER_MAX** }

Functions

- void SoftTimer_Register (eSoftTimerID_t id, uint32_t timeOut, bool IsPeriodic, pfCallBack_t callbackFunction)
- void SoftTimer_Init ()
- void SoftTimer_DeInit ()
- void SoftTimer_Start (eSoftTimerID_t id, bool IsStartNeeded)
- void SoftTimer_Pause (eSoftTimerID_t id, bool IsPauseNeeded)
- void SoftTimer_DelayMS (uint32_t delayMS)
- void SoftTimer_AperiodicTimerSet (uint32_t timeOut)
- bool SoftTimer_HasAperiodicTimerExpired ()
- void SoftTimer_cbPeriodicCheck ()

5.18.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.2

Date

2023-05-28

Copyright

5.18.2 Enumeration Type Documentation

5.18.2.1 eSoftTimerID_t

```
enum eSoftTimerID_t
```

soft-timer IDs

Enumerator

eGENERIC_COUNT_DOWN_TIMER	Generic timer for timeout checks

5.18.3 Function Documentation

5.18.3.1 SoftTimer_Register()

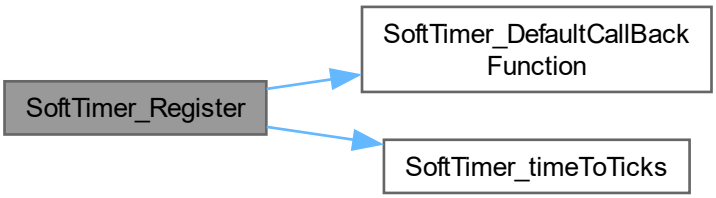
```
void SoftTimer_Register (
            eSoftTimerID_t id,
            uint32_t timeOut,
            bool IsPeriodic,
            pfCallBack_t callbackFunction )
```

Register a periodic function with soft-timer, create ID under eSoftTimerID and attach call-back and time period through this function.

Parameters

id	
timeOut	
IsPeriodic	
callbackFunction	

Here is the call graph for this function:



Here is the caller graph for this function:

5.18.3.2 SoftTimer_Init()

```
void SoftTimer_Init ( )
```

Initialize soft-timer.

Here is the call graph for this function:

Here is the caller graph for this function:

5.18.3.3 SoftTimer_DeInit()

```
void SoftTimer_DeInit ( )
```

DeInitialize soft-timer.

5.18.3.4 SoftTimer_Start()

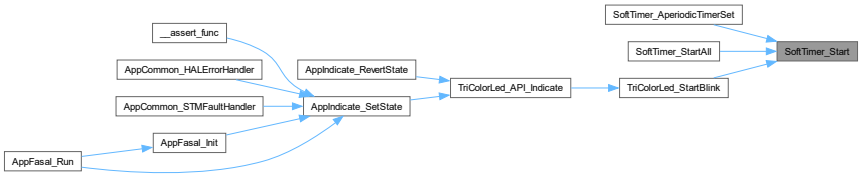
```
void SoftTimer_Start (
            eSoftTimerID_t id,
            bool IsStartNeeded )
```

Start soft-timer.

Parameters

id	
IsStartNeeded	

Here is the caller graph for this function:



5.18.3.5 SoftTimer_Pause()

```
void SoftTimer_Pause (
            eSoftTimerID_t id,
            bool IsPauseNeeded )
```

Function to disable and enable soft-timer for creating critical sections.

Parameters

id	ID of the soft-timer to pause/resume
IsPauseNeeded	Pause if true, resume if false

5.18.3.6 SoftTimer_DelayMS()

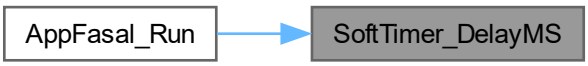
```
void SoftTimer_DelayMS (
            uint32_t delayMS )
```

Wrapper around HAL_Delay delay.

Parameters

delayMS	delay needed in mS

Here is the caller graph for this function:



5.18.3.7 SoftTimer_AperiodicTimerSet()

```
void SoftTimer_AperiodicTimerSet (
            uint32_t timeOut )
```

Set Aperiodic timer instance useful for setting timeouts.

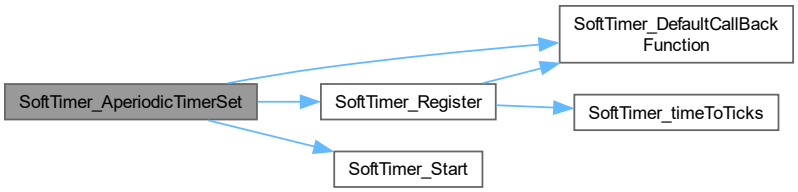
Note

eGENERIC_COUNT_DOWN_TIMER must not be used for other periodic timers

Parameters

timeOut	time out to set the aperiodic timer

Here is the call graph for this function:



5.18.3.8 SoftTimer_HasAperiodicTimerExpired()

```
bool SoftTimer_HasAperiodicTimerExpired ( )
```

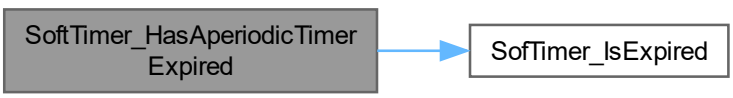
Check if aperiodic timer has expired.

Returns

true if timer has expired

false if timer has not expired/ Registered / started

Here is the call graph for this function:



5.18.3.9 SoftTimer_cbPeriodicCheck()

```
void SoftTimer_cbPeriodicCheck ( )
```

Periodic call from soft-timer that checks if any registered timers have expired and invokes the registered call back HAL_TIM_PeriodElapsedCallback.

Here is the caller graph for this function:

5.19 SoftTimer.h

Go to the documentation of this file.

```
00001
00013
00014 #ifndef UTILITY_SOFTTIMER_SOFTTIMER_H_
00015 #define UTILITY_SOFTTIMER_SOFTTIMER_H_
00016
00018
00019 #include <stdint.h>
00020 #include <stdbool.h>
00021
00022 #include "tim.h"
00023
00025
00026 #define SOFTTIMER_TIMER_INSTANCE        (&htim6)
00027 #define SOFTTIMER_IRQ                   (TIM6_IRQn)
00028 #define SOFTTIMER_OVERFLOW_PERIOD_MS    (10)
00029
00031
00032 typedef void (*pfCallBack_t)(void);
00033
00035
00040 typedef enum
00041 {
00042     eGENERIC_COUNT_DOWN_TIMER,
00043     ePUSH_BUTTON_TIMER,
00044     eDEBUG_LED_SOFT_TIMER,
00045     eSOFT_TIMER_MAX
00046 }eSoftTimerID_t;
00047
00052 typedef struct
00053 {
00054     bool IsArmed;
00055     bool IsPeriodic;
00056     uint32_t currentTicks;
00057     uint32_t setTicks;
00058     pfCallBack_t pfCallBack;
00059 }sSoftTimer_t;
00060
00062
00063 void SoftTimer_Register(eSoftTimerID_t id, uint32_t timeOut, bool IsPeriodic, pfCallBack_t
      callbackFunction );
00064 void SoftTimer_Init();
00065 void SoftTimer_DeInit();
00066 void SoftTimer_Start(eSoftTimerID_t id, bool IsStartNeeded);
00067 void SoftTimer_Pause(eSoftTimerID_t id, bool IsPauseNeeded);
00068 void SoftTimer_DelayMS(uint32_t delayMS);
00069
00070 void SoftTimer_AperiodicTimerSet(uint32_t timeOut);
00071 bool SoftTimer_HasAperiodicTimerExpired();
00072
00073 void SoftTimer_cbPeriodicCheck();
00074
00075
00076
00078
00079
00080 #endif /* UTILITY_SOFTTIMER_SOFTTIMER_H_ */
```

5.20 Utility.h File Reference

```
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
```

Macros

- #define **BUFFER_SIZE_4** (4u)
- #define **BUFFER_SIZE_8** (8u)
- #define **BUFFER_SIZE_16** (16u)
- #define **BUFFER_SIZE_32** (32u)
- #define **BUFFER_SIZE_64** (64u)
- #define **BUFFER_SIZE_128** (128u)
- #define **BUFFER_SIZE_256** (256u)
- #define **BUFFER_SIZE_512** (512u)
- #define **BUFFER_SIZE_1024** (1024u)

5.20.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-05-24

Copyright

Copyright (c) 2023

5.21 Utility.h

Go to the documentation of this file.

```
00001
00012 #ifndef UTILITY_UTILITY_H_
00013 #define UTILITY_UTILITY_H_
00014
00016
00017 #include <stdint.h>
00018 #include <stdbool.h>
00019 #include <stddef.h>
00020
00022
00023 #define BUFFER_SIZE_4           (4u)
00024 #define BUFFER_SIZE_8           (8u)
00025 #define BUFFER_SIZE_16          (16u)
00026 #define BUFFER_SIZE_32          (32u)
00027 #define BUFFER_SIZE_64          (64u)
00028 #define BUFFER_SIZE_128         (128u)
00029 #define BUFFER_SIZE_256         (256u)
00030 #define BUFFER_SIZE_512         (512u)
00031 #define BUFFER_SIZE_1024        (1024u)
00032
00034
00035
00036 #endif /* UTILITY_UTILITY_H_ */
```

5.22 CommonInterrupts.c File Reference

```
#include ¨usart.h¨
#include ¨Console.h¨
#include ¨softTimer.h¨
#include ¨CommonInterrupts.h¨
```

Functions

- void HAL_UART_RxCpltCallback (UART_HandleTypeDef ∗huart)
- void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef ∗htim)

5.22.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-05-28

Copyright

Copyright (c) 2023

5.22.2 Function Documentation

5.22.2.1 HAL_UART_RxCpltCallback()

```
void HAL_UART_RxCpltCallback (
            UART_HandleTypeDef * huart )
```

STM HAL reception callback.

Parameters

huart	

Here is the call graph for this function:

5.22.2.2 HAL_TIM_PeriodElapsedCallback()

```
void HAL_TIM_PeriodElapsedCallback (
            TIM_HandleTypeDef * htim )
```

Timer callback associated with soft-timer, part of STM HAL.

Parameters

htim	timer instance

Here is the call graph for this function:

5.23 CommonInterrupts.h

```
00001 /*
00002  * CommonInterrupts.h
00003  *
00004  *  Created on: Jun 5, 2024
00005  *      Author: Fasal
00006  */
00007
00008 #ifndef COMMONINTERRUPTS_H_
00009 #define COMMONINTERRUPTS_H_
00010
00011
00012
00013 #endif /* COMMONINTERRUPTS_H_ */
```

5.24 ConfigSetting.c File Reference

```
#include ¨GPIO.h¨
#include ¨ConfigSetting.h¨
```

Functions

- eConfigSettingMode_t ConfigSetting_GetCurrentSetting ()

Variables

- static const eConfigSettingMode_t gcConfigSettingHelper [2][2]

5.24.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2024-04-04

Copyright

5.24.2 Function Documentation

5.24.2.1 ConfigSetting_GetCurrentSetting()

```
eConfigSettingMode_t ConfigSetting_GetCurrentSetting ( )
```

Get the existing config Setting.

Returns

eConfigSettingMode_t current config setting

5.24.3 Variable Documentation

5.24.3.1 gcConfigSettingHelper

```
const eConfigSettingMode_t gcConfigSettingHelper[2][2]  [static]
```

Initial value:

```
=
{
        [GPIO_PIN_RESET][GPIO_PIN_RESET]    = eCONFIG_SETTING_0,
        [GPIO_PIN_RESET][GPIO_PIN_SET]      = eCONFIG_SETTING_1,
        [GPIO_PIN_SET]  [GPIO_PIN_RESET]    = eCONFIG_SETTING_2,
        [GPIO_PIN_SET]  [GPIO_PIN_SET]      = eCONFIG_SETTING_3,
}
```

Utility table that maps GPIO levels with eConfigSettingMode_t.

5.25 ConfigSetting.h File Reference

Enumerations

- enum eConfigSettingMode_t {
  eCONFIG_SETTING_0 , **eCONFIG_SETTING_1** , **eCONFIG_SETTING_2** , **eCONFIG_SETTING_3** ,
  eCONFIG_SETTING_MAX }

Functions

- eConfigSettingMode_t ConfigSetting_GetCurrentSetting ()

5.25.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2024-04-04

Copyright

Copyright (c) 2024

5.25.2 Function Documentation

5.25.2.1 ConfigSetting_GetCurrentSetting()

```
eConfigSettingMode_t ConfigSetting_GetCurrentSetting ( )
```

Get the existing config Setting.

Returns

eConfigSettingMode_t current config setting

5.26 ConfigSetting.h

Go to the documentation of this file.

```
00001
00013
00014 #ifndef APPCOMMON_CONFIGSETTING_CONFIGSETTING_H_
00015 #define APPCOMMON_CONFIGSETTING_CONFIGSETTING_H_
00016
00018
00022 typedef enum
00023 {
00024     eCONFIG_SETTING_0,
00025     eCONFIG_SETTING_1,
00026     eCONFIG_SETTING_2,
00027     eCONFIG_SETTING_3,
00028     eCONFIG_SETTING_MAX,
00029 }eConfigSettingMode_t;
00030
00032
00033 eConfigSettingMode_t ConfigSetting_GetCurrentSetting();
00034
00036
00037
00038 #endif /* APPCOMMON_CONFIGSETTING_CONFIGSETTING_H_ */
```

5.27 Console.c File Reference

```
#include <string.h>
#include <stdio.h>
#include <stdarg.h>
#include ¨usart.h¨
#include ¨Console.h¨
#include ¨SoftTimer.h¨
#include ¨AppConfiguration.h¨
```

Functions

- void Console_cbCommandReceived ()
- void Console_Init ()
- void Console_DeInit ()
- eConsolePrintStatus_t Console_TransmitChar (uint8_t data)
- eConsolePrintStatus_t Console_Print (eConsolePrintLevel_t currentLevel, char ∗format,...)
- eConsolePrintStatus_t Console_receive (uint8_t ∗pOutdata, uint16_t length)
- void Console_PrintProgressBar ()
- bool Console_IsCommandRaised (eConsoleCommandsEnum_t cmd)
- void Console_RaiseConsoleCmdRequest (eConsoleCommandsEnum_t cmd)
- void Console_Sync ()

Variables

- static char gDataBuffer [CONSOLE_BUFFER_SIZE] = {0}
- static sConsoleCommand_t gConsoleCommandHelperTable [eCONSOLE_MAX_COMMANDS]
- static volatile sElevatedPromptData_t gvElevatedPromptData = {.buf = {0}}

5.27.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-05-24

Copyright

Copyright (c) 2023

5.27.2 Function Documentation

5.27.2.1 Console_cbCommandReceived()

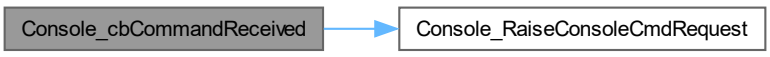
```
void Console_cbCommandReceived ( )
```

Call back function invoked by HAL_UART_RxCpltCallback upon receiving CONSOLE_COMMAND_TOKEN_SIZE characters.

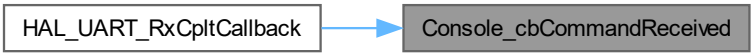
Note

CONSOLE_COMMAND_TOKEN_SIZE is set to strlen(gConsoleCommandHelperTable[eCONSOLE_↩ LEVEL2_ENABLE].CommandStr) when expecting Secret key for level 2 logs

< Check if required passkey is received, if not continue to listen on the console RxHere is the call graph for this function:



Here is the caller graph for this function:



5.27.2.2 Console_Init()

```
void Console_Init ( )
```

Initialize console.

Here is the caller graph for this function:

5.27.2.3 Console_DeInit()

```
void Console_DeInit ( )
```

DeInitialize console.

5.27.2.4 Console_TransmitChar()

```
eConsolePrintStatus_t Console_TransmitChar (
            uint8_t data )
```

Transmit character over Console.

Parameters

data	char to be transmitted

Returns

eConsoleStatus_t

Here is the caller graph for this function:

5.27.2.5 Console_Print()

```
eConsolePrintStatus_t Console_Print (
            eConsolePrintLevel_t currentLevel,
            char * format,
             ...  )
```

Console print function.

Parameters

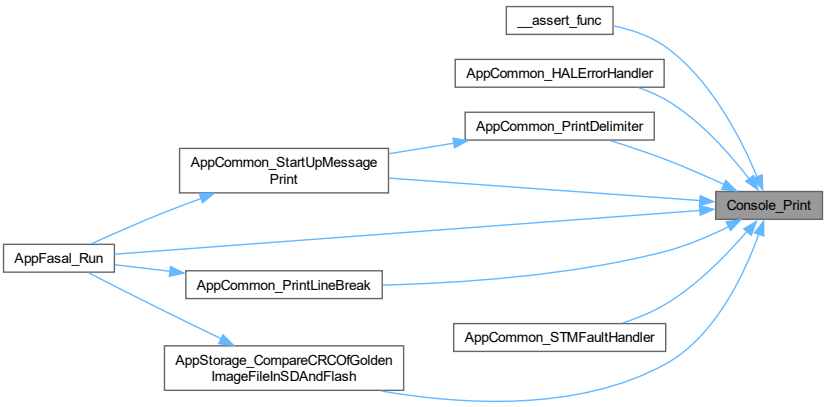
currentLevel	: What level to print the logs in
format	format string
...	

Returns

eConsolePrintStatus_t

< not supported in this Hardware

< For print to get through, print level should be sufficient, Hardware override works on levels below eCONSOLE↩ _PRINT_LVL2Here is the caller graph for this function:



5.27.2.6 Console_receive()

```
eConsolePrintStatus_t Console_receive (
            uint8_t * pOutdata,
            uint16_t length )
```

Receive data over console.

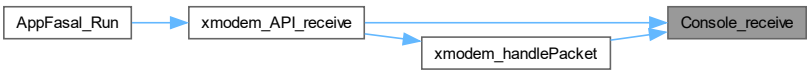
Parameters

pOutdata	Received data is saved here
length	Size of data read

Returns

eConsoleStatus_t

Here is the caller graph for this function:



5.27.2.7 Console_PrintProgressBar()

```
void Console_PrintProgressBar ( )
```

Thin wrapper around UART transmit for Progress Bar print on console.

Here is the caller graph for this function:

5.27.2.8 Console_IsCommandRaised()

```
bool Console_IsCommandRaised (
            eConsoleCommandsEnum_t cmd )
```

Utility function to check if any command request has been made.

Parameters

cmd	

Returns

true

false

5.27.2.9 Console_RaiseConsoleCmdRequest()

```
void Console_RaiseConsoleCmdRequest (
            eConsoleCommandsEnum_t cmd )
```

Raise a request to process a console command.

Here is the caller graph for this function:

5.27.2.10 Console_Sync()

```
void Console_Sync ( )
```

Deferred processing of commands received over console.

5.27.3 Variable Documentation

5.27.3.1 gDataBuffer

```
char gDataBuffer[CONSOLE_BUFFER_SIZE] = {0}  [static]
```

Global buffer used by Console print

5.27.3.2 gConsoleCommandHelperTable

```
sConsoleCommand_t gConsoleCommandHelperTable[eCONSOLE_MAX_COMMANDS]  [static]
```

Initial value:

```
=
{
}
```

Console commands, names and their flags are stored in this table.

5.27.3.3 gvElevatedPromptData

```
volatile sElevatedPromptData_t gvElevatedPromptData = {.buf = {0}}  [static]
```

Buffer for saving elevated prompt data

5.28 Console.h File Reference

```
#include <stdbool.h>
#include ¨Utility.h¨
```

Data Structures

- struct sElevatedPromptData_t
- struct sConsoleCommand_t

Macros

- #define CONSOLE_BUFFER_SIZE (512u)
- #define **CONSOLE_UART_TIMEOUT_MS** (1000u)
- #define **CONSOLE_PROMPT_BUFFER_SIZE** (8u)
- #define **CONSOLE_COMMAND_TOKEN_SIZE** (3u)
- #define **CONSOLE_UART_HANDLE** (&huart1)

Typedefs

- typedef void(∗ pfConsoleCommandActor_t) (void)

Enumerations

- enum eConsolePrintStatus_t { **eCONSOLE_SUCCESS** , **eCONSOLE_FAIL** , **eCONSOLE_STATUS_MAX** }
- enum eConsoleCommandsEnum_t { **eCONSOLE_LEVEL1_ENABLE** , **eCONSOLE_LEVEL2_REQUEST** , **eCONSOLE_LEVEL2_ENABLE** , **eCONSOLE_ERASE_FLASH_REQUEST** , **eCONSOLE_SENSOR_TEST_REQUEST** , **eCONSOLE_MAX_COMMANDS** }
- enum eConsolePrintLevel_t { **eCONSOLE_PRINT_LVL0** , **eCONSOLE_PRINT_LVL1** , **eCONSOLE_↩ PRINT_LVL2** , **eCONSOLE_PRINT_LVL_MAX** }

Functions

- void Console_Init ()
- void Console_DeInit ()
- eConsolePrintStatus_t Console_TransmitChar (uint8_t data)
- eConsolePrintStatus_t Console_Print (eConsolePrintLevel_t currentLevel, char ∗format,...)
- eConsolePrintStatus_t Console_receive (uint8_t ∗pOutdata, uint16_t length)
- void Console_cbCommandReceived ()
- bool Console_IsCommandRaised (eConsoleCommandsEnum_t cmd)
- void Console_RaiseConsoleCmdRequest (eConsoleCommandsEnum_t cmd)
- void Console_Sync ()
- void Console_PrintProgressBar ()

5.28.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-05-24

Copyright

5.28.2 Macro Definition Documentation

5.28.2.1 CONSOLE_BUFFER_SIZE

```
#define CONSOLE_BUFFER_SIZE (512u)
```

Console buffer size, To be increased if larger strings are to be printed.

Maximum expected response is just under 200 bytes

5.28.3 Typedef Documentation

5.28.3.1 pfConsoleCommandActor_t

```
typedef void(* pfConsoleCommandActor_t) (void)
```

Console command handlers.

5.28.4 Enumeration Type Documentation

5.28.4.1 eConsolePrintStatus_t

```
enum eConsolePrintStatus_t
```

Enum that maintains Console status.

5.28.4.2 eConsoleCommandsEnum_t

```
enum eConsoleCommandsEnum_t
```

Command names are enumerated here.

5.28.4.3 eConsolePrintLevel_t

```
enum eConsolePrintLevel_t
```

Console print level.

5.28.5 Function Documentation

5.28.5.1 Console_Init()

```
void Console_Init ( )
```

Initialize console.

Here is the caller graph for this function:

5.28.5.2 Console_DeInit()

```
void Console_DeInit ( )
```

DeInitialize console.

5.28.5.3 Console_TransmitChar()

```
eConsolePrintStatus_t Console_TransmitChar (
            uint8_t data )
```

Transmit character over Console.

Parameters

data	char to be transmitted

Returns

eConsoleStatus_t

Here is the caller graph for this function:

5.28.5.4 Console_Print()

```
eConsolePrintStatus_t Console_Print (
            eConsolePrintLevel_t currentLevel,
            char * format,
             ...  )
```

Console print function.

Parameters

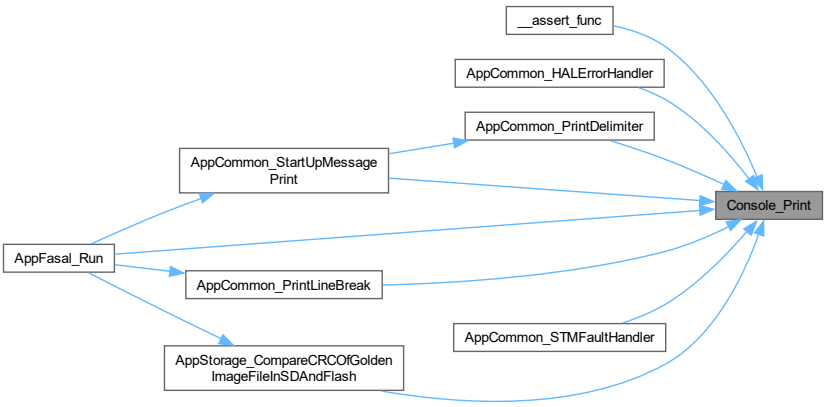
currentLevel	: What level to print the logs in
format	format string
...	

Returns

eConsolePrintStatus_t

< not supported in this Hardware

< For print to get through, print level should be sufficient, Hardware override works on levels below eCONSOLE↩ _PRINT_LVL2Here is the caller graph for this function:



5.28.5.5 Console_receive()

```
eConsolePrintStatus_t Console_receive (
            uint8_t * pOutdata,
            uint16_t length )
```

Receive data over console.

Parameters

pOutdata	Received data is saved here
length	Size of data read

Returns

eConsoleStatus_t

Here is the caller graph for this function:

5.28.5.6 Console_cbCommandReceived()

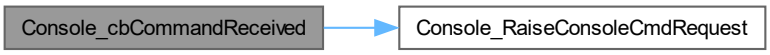
```
void Console_cbCommandReceived ( )
```

Call back function invoked by HAL_UART_RxCpltCallback upon receiving CONSOLE_COMMAND_TOKEN_SIZE characters.

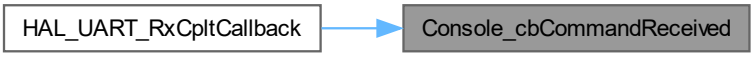
Note

CONSOLE_COMMAND_TOKEN_SIZE is set to strlen(gConsoleCommandHelperTable[eCONSOLE_↩ LEVEL2_ENABLE].CommandStr) when expecting Secret key for level 2 logs

< Check if required passkey is received, if not continue to listen on the console RxHere is the call graph for this function:



Here is the caller graph for this function:



5.28.5.7 Console_IsCommandRaised()

```
bool Console_IsCommandRaised (
            eConsoleCommandsEnum_t cmd )
```

Utility function to check if any command request has been made.

Parameters

cmd	

Returns

true

false

5.28.5.8 Console_RaiseConsoleCmdRequest()

```
void Console_RaiseConsoleCmdRequest (
            eConsoleCommandsEnum_t cmd )
```

Raise a request to process a console command.

Here is the caller graph for this function:

5.28.5.9 Console_Sync()

```
void Console_Sync ( )
```

Deferred processing of commands received over console.

5.28.5.10 Console_PrintProgressBar()

```
void Console_PrintProgressBar ( )
```

Thin wrapper around UART transmit for Progress Bar print on console.

Here is the caller graph for this function:

5.29 Console.h

Go to the documentation of this file.

```
00001
00013
00014 #ifndef CONSOLE_CONSOLE_H_
00015 #define CONSOLE_CONSOLE_H_
00016
00018
00019 #include <stdbool.h>
00020 #include "Utility.h"
00021
00023
00024
00029 #define CONSOLE_BUFFER_SIZE         (512u)
00030 #define CONSOLE_UART_TIMEOUT_MS     (1000u)
00031 #define CONSOLE_PROMPT_BUFFER_SIZE  (8u)
00032 #define CONSOLE_COMMAND_TOKEN_SIZE  (3u)
00033
00034 #define CONSOLE_UART_HANDLE         (&huart1)
00035
00037
00042 typedef void (*pfConsoleCommandActor_t)(void);
00043
00048 typedef enum
00049 {
00050     eCONSOLE_SUCCESS,
00051     eCONSOLE_FAIL,
00052     eCONSOLE_STATUS_MAX
00053 }eConsolePrintStatus_t;
00054
00059 typedef enum
00060 {
00061     eCONSOLE_LEVEL1_ENABLE,
00062     eCONSOLE_LEVEL2_REQUEST,
00063     eCONSOLE_LEVEL2_ENABLE,
00064     eCONSOLE_ERASE_FLASH_REQUEST,
00065     eCONSOLE_SENSOR_TEST_REQUEST,
00066     eCONSOLE_MAX_COMMANDS
00067 }eConsoleCommandsEnum_t;
00068
00073 typedef enum
00074 {
00075     eCONSOLE_PRINT_LVL0,
00076     eCONSOLE_PRINT_LVL1,
00077     eCONSOLE_PRINT_LVL2,
00078     eCONSOLE_PRINT_LVL_MAX,
00079 }eConsolePrintLevel_t;
00080
00082
00087 typedef struct
00088 {
00089     char buf[CONSOLE_PROMPT_BUFFER_SIZE];
00090 }sElevatedPromptData_t;
00091
00096 typedef struct
00097 {
00098     const char* CommandName;
00099     char CommandStr[BUFFER_SIZE_8];
00100     volatile bool IsCommandRaised;
00101     volatile bool IsCommandServiced;
00102     pfConsoleCommandActor_t pfConsoleCommandActor;
00103 }sConsoleCommand_t;
00104
00106
00107 void Console_Init();
00108 void Console_DeInit();
00109 eConsolePrintStatus_t Console_TransmitChar(uint8_t data);
00110 eConsolePrintStatus_t Console_Print(eConsolePrintLevel_t currentLevel, char *format,...);
00111 eConsolePrintStatus_t Console_receive(uint8_t* pOutdata, uint16_t length);
00112 void Console_cbCommandReceived();
00113 bool Console_IsCommandRaised(eConsoleCommandsEnum_t cmd);
00114 void Console_RaiseConsoleCmdRequest(eConsoleCommandsEnum_t cmd);
00115 void Console_Sync();
00116 void Console_PrintProgressBar();
00117
00119
00120 #endif /* CONSOLE_CONSOLE_H_ */
```

5.30 PushButton.c File Reference

```
#include <stdint.h>
#include <stdbool.h>
#include ¨gpio.h¨
#include ¨PushButton.h¨
```

Functions

- bool PushButton_IsFlashButtonPressed ()

5.30.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2024-04-04

Copyright

Copyright (c) 2024

5.30.2 Function Documentation

5.30.2.1 PushButton_IsFlashButtonPressed()

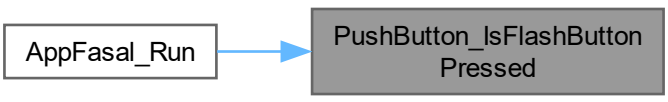
```
bool PushButton_IsFlashButtonPressed ( )
```

Check if Push Button is pressed.

Returns

true if button is pressed, false if not during function invocation

Here is the caller graph for this function:



5.31 PushButton.h File Reference

```
#include <stdint.h>
#include <stdbool.h>
#include ¨gpio.h¨
#include ¨AppConfiguration.h¨
```

Functions

- bool PushButton_IsFlashButtonPressed ()

5.31.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2024-04-04

Copyright

5.31.2 Function Documentation

5.31.2.1 PushButton_IsFlashButtonPressed()

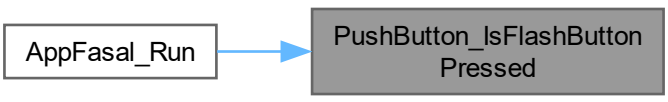
```
bool PushButton_IsFlashButtonPressed ( )
```

Check if Push Button is pressed.

Returns

true if button is pressed, false if not during function invocation

Here is the caller graph for this function:



5.32 PushButton.h

Go to the documentation of this file.

```
00001
00013
00014 #ifndef APPCOMMON_PUSHBUTTON_PUSHBUTTON_H_
00015 #define APPCOMMON_PUSHBUTTON_PUSHBUTTON_H_
00016
00018
00019 #include <stdint.h>
00020 #include <stdbool.h>
00021 #include "gpio.h"
00022 #include "AppConfiguration.h"
00023
00025
00026 bool PushButton_IsFlashButtonPressed();
00027
00029
00030 #endif /* APPCOMMON_PUSHBUTTON_PUSHBUTTON_H_ */
```

5.33 TriColorLED.c File Reference

```
#include <assert.h>
#include ¨TriColorLED.h¨
#include ¨SoftTimer.h¨
```

Functions

- static void TriColorLed_ToggleState (volatile sTricolorLED_t ∗const pMe)
- static volatile sTricolorLED_t ∗const TriColorLed_GetInstance ()
- static void TriColorLED_cbBlinkHandler ()
- static void TriColorLed_SetState (volatile sTricolorLED_t ∗const pMe, eTriColorLEDStates_t triColor↩ LEDState)
- static void TriColorLed_Init (volatile sTricolorLED_t ∗const pMe)
- static void TriColorLed_DeInit (volatile sTricolorLED_t ∗const pMe)
- static void TriColorLed_StartBlink (volatile sTricolorLED_t ∗const pMe, eTriColorLEDBlinkPeriod_t eBlink↩ Period, bool IsStartNeeded)
- void TriColorLed_API_Init ()
- void TriColorLed_API_DeInit ()
- void TriColorLed_API_SetState (eTriColorLEDStates_t state)
- void TriColorLed_API_Indicate (eTriColorLEDStates_t state, eTriColorLEDBlinkPeriod_t eBlinkPeriod, bool IsBlinkNeeded)

Variables

- static const GPIO_PinState gcLEDEnumtoGPIOStateConverter [eLED_STATE_MAX]
- static const sLEDPins_t gcLEDPins [eLED_MAX]
- static const sLEDPins_t gcLEDPins2 [eLED_MAX]
- static const sTriColorLEDState_t gcLEDStateHelperTable [eLED_COLOR_STATE_MAX]
- static const uint32_t gcBlinkPeriodToCountHelper [eLED_BLINK_PERIOD_MAX]
- static volatile sTricolorLED_t gvTricolorLed

5.33.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-09-28

Copyright

5.33.2 Function Documentation

5.33.2.1 TriColorLed_ToggleState()

```
static void TriColorLed_ToggleState (
            volatile sTricolorLED_t *const pMe )  [static]
```

Toggle TriColor LED.

Parameters

pMe	TriColor LED instance

Here is the call graph for this function:

Here is the caller graph for this function:

5.33.2.2 TriColorLed_GetInstance()

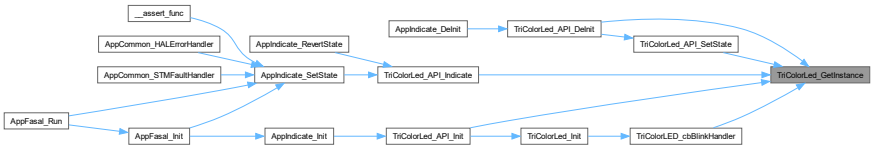
```
static volatile sTricolorLED_t *const TriColorLed_GetInstance ( )  [static]
```

Get tricolor LED instance.

Returns

volatile* const

Here is the caller graph for this function:

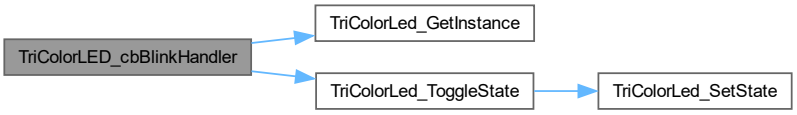


5.33.2.3 TriColorLED_cbBlinkHandler()

```
static void TriColorLED_cbBlinkHandler ( )  [static]
```

Call back function registered with softtimer.

Here is the call graph for this function:



Here is the caller graph for this function:

5.33.2.4 TriColorLed_SetState()

```
static void TriColorLed_SetState (
            volatile sTricolorLED_t *const pMe,
            eTriColorLEDStates_t triColorLEDState )  [static]
```

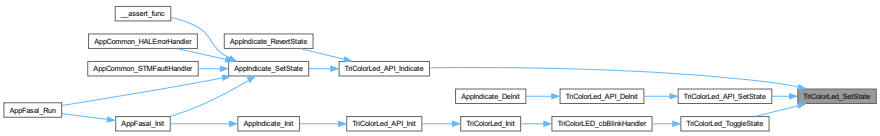
Set TriColor LED state.

Parameters

pMe	TriColor LED instance
triColorLEDState	

< Set Primary set of LED

< Duplicate set of LED follows primary LED indicationHere is the caller graph for this function:



5.33.2.5 TriColorLed_Init()

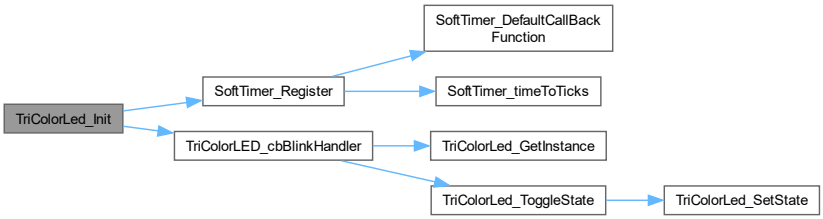
```
static void TriColorLed_Init (
            volatile sTricolorLED_t *const pMe )  [static]
```

Initialize Tricolor LED.

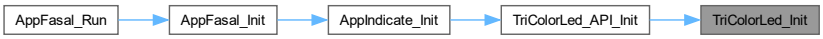
Parameters

pMe	TriColor LED instance

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.2.6 TriColorLed_DeInit()

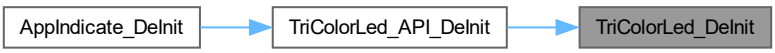
```
static void TriColorLed_DeInit (
            volatile sTricolorLED_t *const pMe )  [static]
```

DeInitialize Tricolor LED.

Parameters

pMe	TriColor LED instance

Here is the caller graph for this function:



5.33.2.7 TriColorLed_StartBlink()

```
static void TriColorLed_StartBlink (
            volatile sTricolorLED_t *const pMe,
            eTriColorLEDBlinkPeriod_t eBlinkPeriod,
            bool IsStartNeeded )  [static]
```

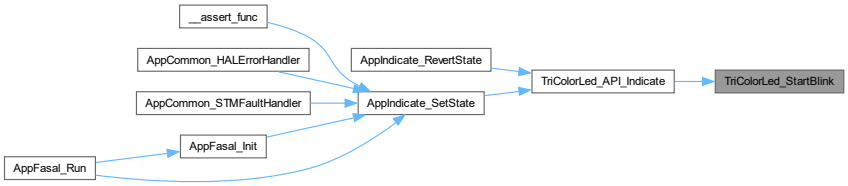
Enable/Disable LED blink routine. The Blink will be executed though TriColorLED_cbBlinkHandler.

Parameters

pMe	TriColor LED instance
eBlinkPeriod	
IsStartNeeded	Pass true if blink needed, false to cancel blink

Here is the call graph for this function:

Here is the caller graph for this function:

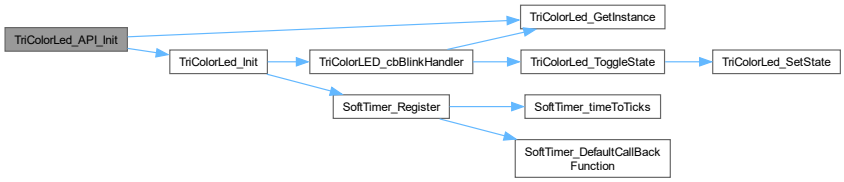


5.33.2.8 TriColorLed_API_Init()

```
void TriColorLed_API_Init ( )
```

Initialize tricolor LED API.

Here is the call graph for this function:



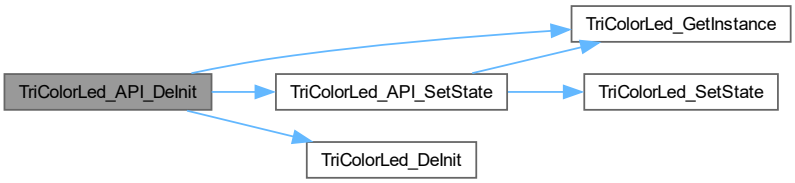
Here is the caller graph for this function:

5.33.2.9 TriColorLed_API_DeInit()

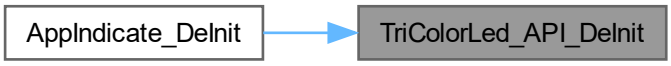
```
void TriColorLed_API_DeInit ( )
```

DeInitialize tricolor LED API.

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.2.10 TriColorLed_API_SetState()

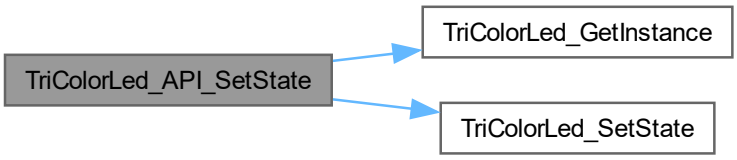
```
void TriColorLed_API_SetState (
            eTriColorLEDStates_t state )
```

API to set Tricolor LED state.

Parameters

state	state to set TriColor LED

Here is the call graph for this function:



Here is the caller graph for this function:

5.33.2.11 TriColorLed_API_Indicate()

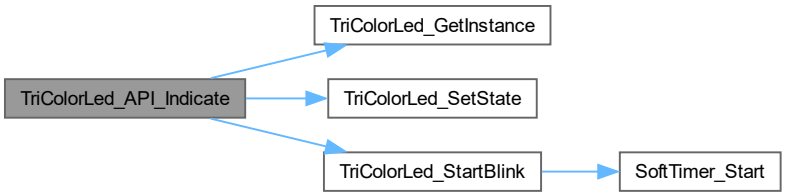
```
void TriColorLed_API_Indicate (
            eTriColorLEDStates_t state,
            eTriColorLEDBlinkPeriod_t eBlinkPeriod,
            bool IsBlinkNeeded )
```

Tricolor LED to indicate various Application states.

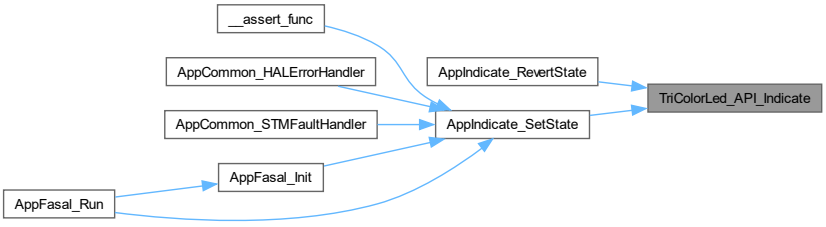
Parameters

state	TriColor LED color to set
eBlinkPeriod	Period to blink LED over
IsBlinkNeeded	Set to true if blinking is needed, else the LED will only light up

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.3 Variable Documentation

5.33.3.1 gcLEDEnumtoGPIOStateConverter

```
const GPIO_PinState gcLEDEnumtoGPIOStateConverter[eLED_STATE_MAX]  [static]
```

Initial value:

```
=
{
        [eLED_OFF] = GPIO_PIN_RESET,
        [eLED_ON] = GPIO_PIN_SET,
}
```

Helper table that maps LED state to GPIO state.

5.33.3.2 gcLEDPins

```
const sLEDPins_t gcLEDPins[eLED_MAX]  [static]
```

Initial value:

```
=
{
        [eLED_RED]      = {.LEDPort = LED_R_GPIO_Port, .LEDPin = LED_R_Pin },
        [eLED_GREEN]    = {.LEDPort = LED_G_GPIO_Port, .LEDPin = LED_G_Pin },
        [eLED_BLUE]     = {.LEDPort = LED_B_GPIO_Port, .LEDPin = LED_B_Pin },
}
```

LED pin mapping helper table.

5.33.3.3 gcLEDPins2

```
const sLEDPins_t gcLEDPins2[eLED_MAX]  [static]
```

Initial value:

```
=
{
        [eLED_BLUE]     = {.LEDPort = LED_R1_GPIO_Port, .LEDPin = LED_R1_Pin },
        [eLED_GREEN]    = {.LEDPort = LED_G1_GPIO_Port, .LEDPin = LED_G1_Pin },
        [eLED_RED]      = {.LEDPort = LED_B1_GPIO_Port, .LEDPin = LED_B1_Pin },
}
```

LED pin mapping helper table.

- warning eLED_BLUE and eLED_RED are swapped in Rev 1A Hardware

5.33.3.4 gcLEDStateHelperTable

```
const sTriColorLEDState_t gcLEDStateHelperTable[eLED_COLOR_STATE_MAX]  [static]
```

Helper structure to map TriColor LED enumerations to Individual LED combinations.

5.33.3.5 gcBlinkPeriodToCountHelper

```
const uint32_t gcBlinkPeriodToCountHelper[eLED_BLINK_PERIOD_MAX]  [static]
```

Initial value:

```
=
{
        [eTRICOLORLED_BLINK_NONE]       = UINT32_MAX,
        [eTRICOLORLED_BLINK_250MS]      = 250/TRICOLOR_LED_BLINK_TIME_BASE_MS,
        [eTRICOLORLED_BLINK_500MS]      = 500/TRICOLOR_LED_BLINK_TIME_BASE_MS,
        [eTRICOLORLED_BLINK_1000MS]     = 1000/TRICOLOR_LED_BLINK_TIME_BASE_MS,
        [eTRICOLORLED_BLINK_2000MS]     = 2000/TRICOLOR_LED_BLINK_TIME_BASE_MS
}
```

Utility table that saves LED timings to ticks based on TRICOLOR_LED_BLINK_TIME_BASE_MS.

5.33.3.6 gvTricolorLed

```
volatile sTricolorLED_t gvTricolorLed  [static]
```

Tricolor LED instance.

5.34 TriColorLED.h File Reference

```
#include <stdint.h>
#include <stddef.h>
#include <stdbool.h>
#include "gpio.h"
```

Data Structures

- struct sTriColorLEDState_t
- struct sLEDPins_t
- struct sLed_t
- struct sTricolorLED_t
- struct sTriColorIndication_t

Macros

- #define **TRICOLOR_LED_BLINK_TIME_BASE_MS** (50u) /** < This is the timebase from which all LED blink period is composed of */

Enumerations

- enum eLEDId_t { **eLED_RED** , **eLED_GREEN** , **eLED_BLUE** , **eLED_MAX** }
- enum eLEDState_t { **eLED_OFF** , **eLED_ON** , **eLED_STATE_MAX** }
- enum eTriColorLEDBlinkPeriod_t {
  eTRICOLORLED_BLINK_NONE , **eTRICOLORLED_BLINK_250MS** , **eTRICOLORLED_BLINK_500MS** ,
  eTRICOLORLED_BLINK_1000MS ,
  eTRICOLORLED_BLINK_2000MS , **eLED_BLINK_PERIOD_MAX** }
- enum eTriColorLEDStates_t {
  eLED_COLOR_ALL_OFF , **eLED_COLOR_RED** , **eLED_COLOR_GREEN** , **eLED_COLOR_BLUE** ,
  eLED_COLOR_YELLOW , **eLED_COLOR_MAGENTA** , **eLED_COLOR_CYAN** , **eLED_COLOR_WHITE** ,
  eLED_COLOR_STATE_MAX }

Functions

- void TriColorLed_API_Init ()
- void TriColorLed_API_DeInit ()
- void TriColorLed_API_SetState (eTriColorLEDStates_t state)
- void TriColorLed_API_Indicate (eTriColorLEDStates_t state, eTriColorLEDBlinkPeriod_t eBlinkPeriod, bool IsBlinkNeeded)

5.34.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-09-28

Copyright

5.34.2 Enumeration Type Documentation

5.34.2.1 eLEDId_t

`enum eLEDId_t`

LEDs that make up tricolor LEDs.

5.34.2.2 eLEDState_t

`enum eLEDState_t`

LED state enumeration.

5.34.2.3 eTriColorLEDBlinkPeriod_t

`enum eTriColorLEDBlinkPeriod_t`

Possible LED blink periods.

5.34.2.4 eTriColorLEDStates_t

```
enum eTriColorLEDStates_t
```

TriColor LED state enumerations.

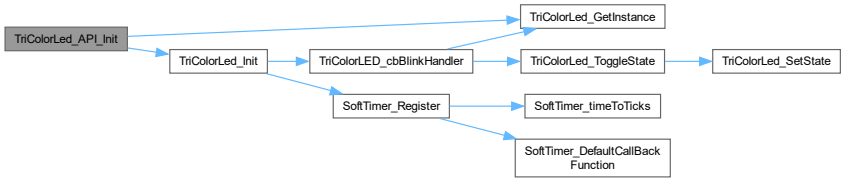
5.34.3 Function Documentation

5.34.3.1 TriColorLed_API_Init()

```
void TriColorLed_API_Init ( )
```

Initialize tricolor LED API.

Here is the call graph for this function:



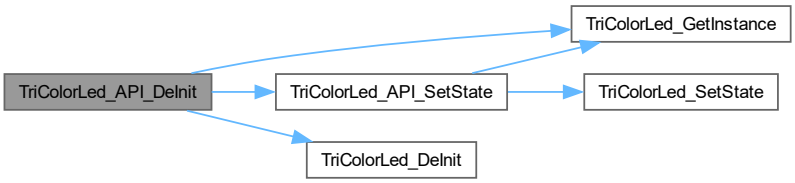
Here is the caller graph for this function:

5.34.3.2 TriColorLed_API_DeInit()

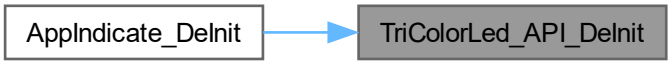
```
void TriColorLed_API_DeInit ( )
```

DeInitialize tricolor LED API.

Here is the call graph for this function:



Here is the caller graph for this function:



5.34.3.3 TriColorLed_API_SetState()

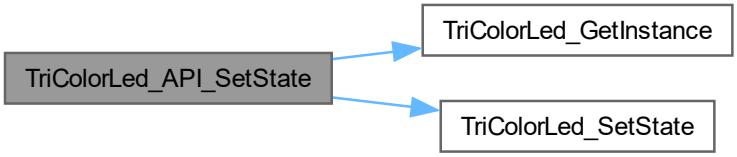
```
void TriColorLed_API_SetState (
            eTriColorLEDStates_t state )
```

API to set Tricolor LED state.

Parameters

state	state to set TriColor LED

Here is the call graph for this function:



Here is the caller graph for this function:

5.34.3.4 TriColorLed_API_Indicate()

```
void TriColorLed_API_Indicate (
            eTriColorLEDStates_t state,
```

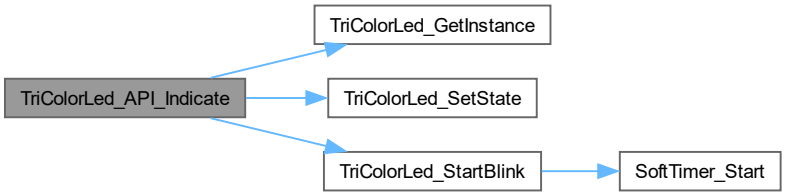
```
            eTriColorLEDBlinkPeriod_t eBlinkPeriod,
            bool IsBlinkNeeded )
```

Tricolor LED to indicate various Application states.

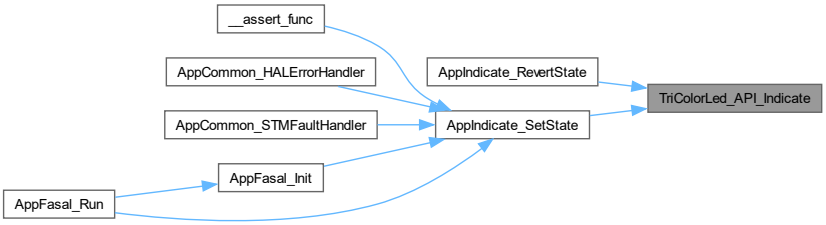
Parameters

state	TriColor LED color to set
eBlinkPeriod	Period to blink LED over
IsBlinkNeeded	Set to true if blinking is needed, else the LED will only light up

Here is the call graph for this function:



Here is the caller graph for this function:



5.35 TriColorLED.h

Go to the documentation of this file.

```
00001
00013
00014 #ifndef APPCOMMON_TRICOLORLED_TRICOLORLED_H_
00015 #define APPCOMMON_TRICOLORLED_TRICOLORLED_H_
00016
00018
00019 #include <stdint.h>
00020 #include <stddef.h>
00021 #include <stdbool.h>
00022 #include "gpio.h"
00023
00025
00026 #define TRICOLOR_LED_BLINK_TIME_BASE_MS (50u)
00029
00034 typedef enum
```

```
00035 {
00036     eLED_RED,
00037     eLED_GREEN,
00038     eLED_BLUE,
00039     eLED_MAX
00040 }eLEDId_t;
00041
00046 typedef enum
00047 {
00048     eLED_OFF,
00049     eLED_ON,
00050     eLED_STATE_MAX
00051 }eLEDState_t;
00052
00057 typedef enum
00058 {
00059     eTRICOLORLED_BLINK_NONE,
00060     eTRICOLORLED_BLINK_250MS,
00061     eTRICOLORLED_BLINK_500MS,
00062     eTRICOLORLED_BLINK_1000MS,
00063     eTRICOLORLED_BLINK_2000MS,
00064     eLED_BLINK_PERIOD_MAX
00065 }eTriColorLEDBlinkPeriod_t;
00066
00071 typedef enum
00072 {
00073     eLED_COLOR_ALL_OFF,
00074     eLED_COLOR_RED,
00075     eLED_COLOR_GREEN,
00076     eLED_COLOR_BLUE,
00077     eLED_COLOR_YELLOW,
00078     eLED_COLOR_MAGENTA,
00079     eLED_COLOR_CYAN,
00080     eLED_COLOR_WHITE,
00081     eLED_COLOR_STATE_MAX
00082 }eTriColorLEDStates_t;
00083
00085
00090 typedef struct
00091 {
00092     eLEDState_t LedState[eLED_MAX];
00093 }sTriColorLEDState_t;
00094
00099 typedef struct
00100 {
00101     GPIO_TypeDef* LEDPort;
00102     uint16_t LEDPin;
00103 }sLEDPins_t;
00104
00109 typedef struct
00110 {
00111     const sLEDPins_t* pcLedPin;
00112     eLEDState_t ledState;
00113 }sLed_t;
00114
00119 typedef struct
00120 {
00121     bool IsInitialized;
00122     bool IsBlinkEnabled;
00123     size_t blinkTicksSet;
00124     size_t blinkTicksCurrent;
00125     eTriColorLEDStates_t triColorLEDState;
00126     eTriColorLEDStates_t prevOnLEDState;
00127     sLed_t Leds[eLED_MAX];
00128     sLed_t Leds2[eLED_MAX];
00129 }sTricolorLED_t;
00130
00135 typedef struct
00136 {
00137     bool IsBlinkNeeded;
00138     eTriColorLEDStates_t ledState;
00139     eTriColorLEDBlinkPeriod_t ledBlinkPeriod;
00140 }sTriColorIndication_t;
00141
00143
00144 void TriColorLed_API_Init();
00145 void TriColorLed_API_DeInit();
00146 void TriColorLed_API_SetState(eTriColorLEDStates_t state);
00147 void TriColorLed_API_Indicate(eTriColorLEDStates_t state, eTriColorLEDBlinkPeriod_t eBlinkPeriod, bool
    IsBlinkNeeded);
00148
00150
00151 #endif /* APPCOMMON_TRICOLORLED_TRICOLORLED_H_ */
```

5.36 AppFasal.c File Reference

```
#include <assert.h>
#include <stdbool.h>
#include "AppFasal.h"
#include "AppCommon.h"
#include "AppIndication.h"
#include "Console.h"
#include "SoftTimer.h"
#include "DebugPrint.h"
#include "AppStorage.h"
#include "PushButton.h"
#include "AppStorageDataStructures.h"
#include "AppFlash_API.h"
#include "AppSD_API.h"
#include "ConfigSetting.h"
#include "xmodem.h"
#include "AppProfiler.h"
#include "AppConfiguration.h"
```

Functions

- static void AppFasal_Init ()
- eAppFasalStates_t AppFasal_Run ()

Variables

- static const eAppIndicationStates_t gcIndicationToAppStateMap [eFASAL_MAX_STATE]

5.36.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-06-23

Copyright

Copyright (c) 2023

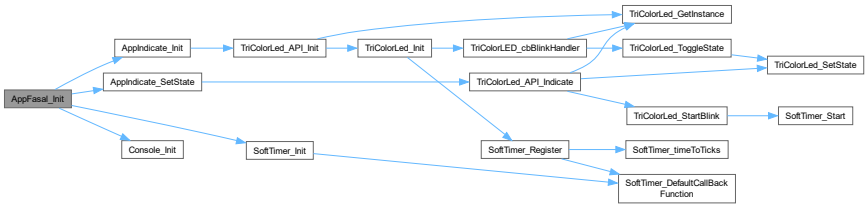
5.36.2 Function Documentation

5.36.2.1 AppFasal_Init()

```
static void AppFasal_Init ( )  [static]
```

Initialize all Application modules here.

Here is the call graph for this function:



Here is the caller graph for this function:

5.36.2.2 AppFasal_Run()

```
eAppFasalStates_t AppFasal_Run ( )
```

Application Run.

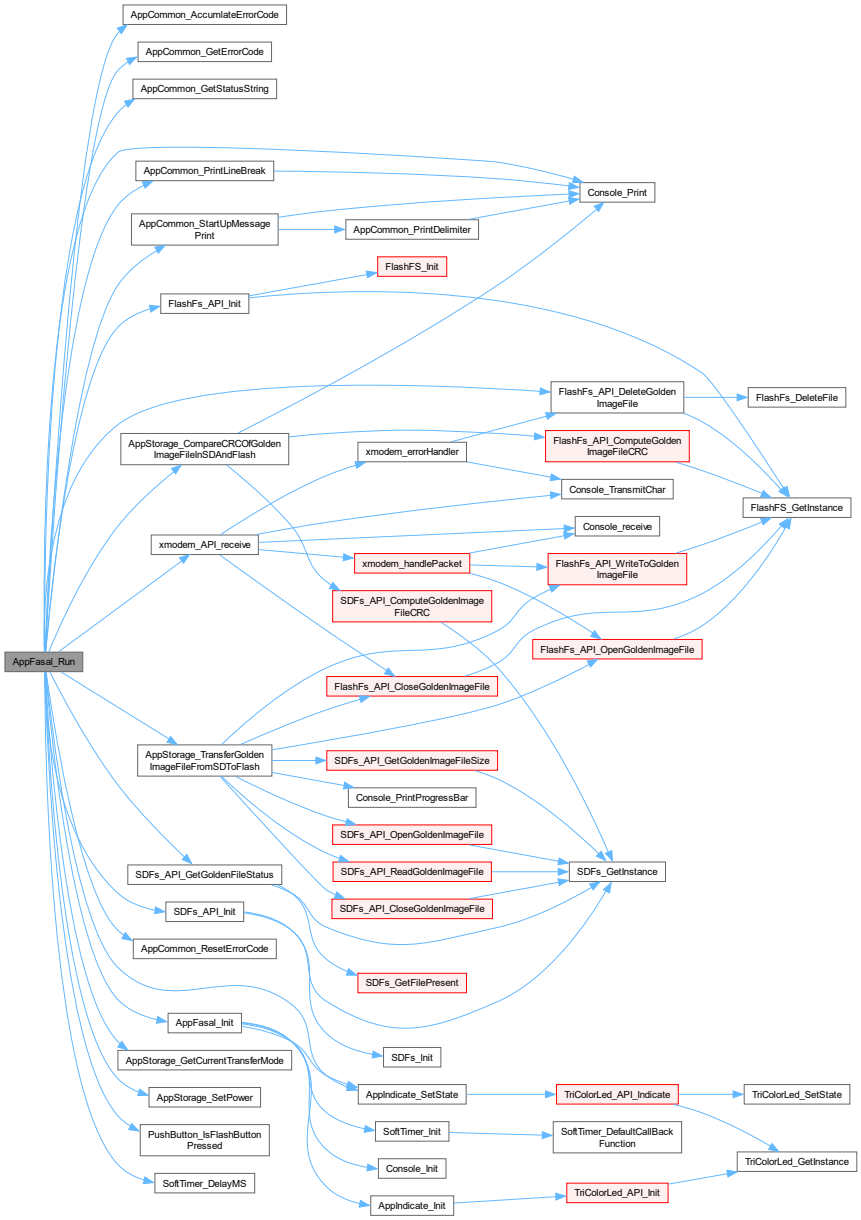
Returns

eAppFasalStates_t

< Set power to External Flash prior to Initializing the same

< Stop powering the external flash since transfer operation is complete

< Errors from previous run if any must be cleared hereHere is the call graph for this function:



5.36.3 Variable Documentation

5.36.3.1 gcIndicationToAppStateMap

```
const eAppIndicationStates_t gcIndicationToAppStateMap[eFASAL_MAX_STATE]  [static]
```

Initial value:

```
=
{
        [eFASAL_APP_INIT]               = eIND_BLUE_0,
        [eFASAL_APP_STARTUP_MSG]        = eIND_BLUE_250MS,
        [eFASAL_APP_BUTTON_WAIT]        = eIND_BLUE_500MS,
        [eFASAL_APP_SD_INIT]            = eIND_BLUE_1000MS,
        [eFASAL_APP_SD_CHECK]           = eIND_BLUE_1000MS,
```

```
        [eFASAL_APP_FLASH_INIT]          = eIND_BLUE_1000MS,
        [eFASAL_APP_MODE_SELECTION]      = eIND_BLUE_1000MS,
        [eFASAL_APP_SD_FLASH_TRANSFER]   = eIND_YELLOW_1000MS,
        [eFASAL_APP_XMODEM_TRANSFER]     = eIND_YELLOW_1000MS,
        [eFASAL_APP_CRC_COMPARE]         = eIND_YELLOW_1000MS,
        [eFASAL_APP_TRANSFER_SUCCESS]    = eIND_GREEN_0,
        [eFASAL_APP_SD_FAIL]             = eIND_RED_250MS,
        [eFASAL_APP_SD_FILE_FAIL]        = eIND_RED_250MS,
        [eFASAL_APP_FLASH_FAIL]          = eIND_RED_250MS,
        [eFASAL_APP_TRANSFER_FAIL]       = eIND_RED_250MS,
        [eFASAL_APP_CRC_FAIL]            = eIND_RED_250MS,
        [eFASAL_APP_END]                 = eIND_NO_CHANGE,
}
```

Map App state with indication state.

5.37 AppFasal.h File Reference

```
#include ¨AppCommon.h¨
```

Enumerations

- enum eAppFasalStates_t {
  eFASAL_APP_INIT , **eFASAL_APP_STARTUP_MSG** , **eFASAL_APP_BUTTON_WAIT** , **eFASAL_APP↩ _SD_INIT** ,
  eFASAL_APP_SD_CHECK , **eFASAL_APP_FLASH_INIT** , **eFASAL_APP_MODE_SELECTION** , **e↩ FASAL_APP_SD_FLASH_TRANSFER** ,
  eFASAL_APP_XMODEM_TRANSFER , **eFASAL_APP_CRC_COMPARE** , **eFASAL_APP_TRANSFER_↩ SUCCESS** , **eFASAL_APP_SD_FAIL** ,
  eFASAL_APP_SD_FILE_FAIL , **eFASAL_APP_FLASH_FAIL** , **eFASAL_APP_TRANSFER_FAIL** , **e↩ FASAL_APP_CRC_FAIL** ,
  eFASAL_APP_END , **eFASAL_MAX_STATE** }

Functions

- eAppFasalStates_t AppFasal_Run ()

5.37.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-06-23

Copyright

5.37.2 Enumeration Type Documentation

5.37.2.1 eAppFasalStates_t

```
enum eAppFasalStates_t
```

Enumeration for various states of AppFasal_RunNova.

5.37.3 Function Documentation

5.37.3.1 AppFasal_Run()

```
eAppFasalStates_t AppFasal_Run ( )
```

Application Run.

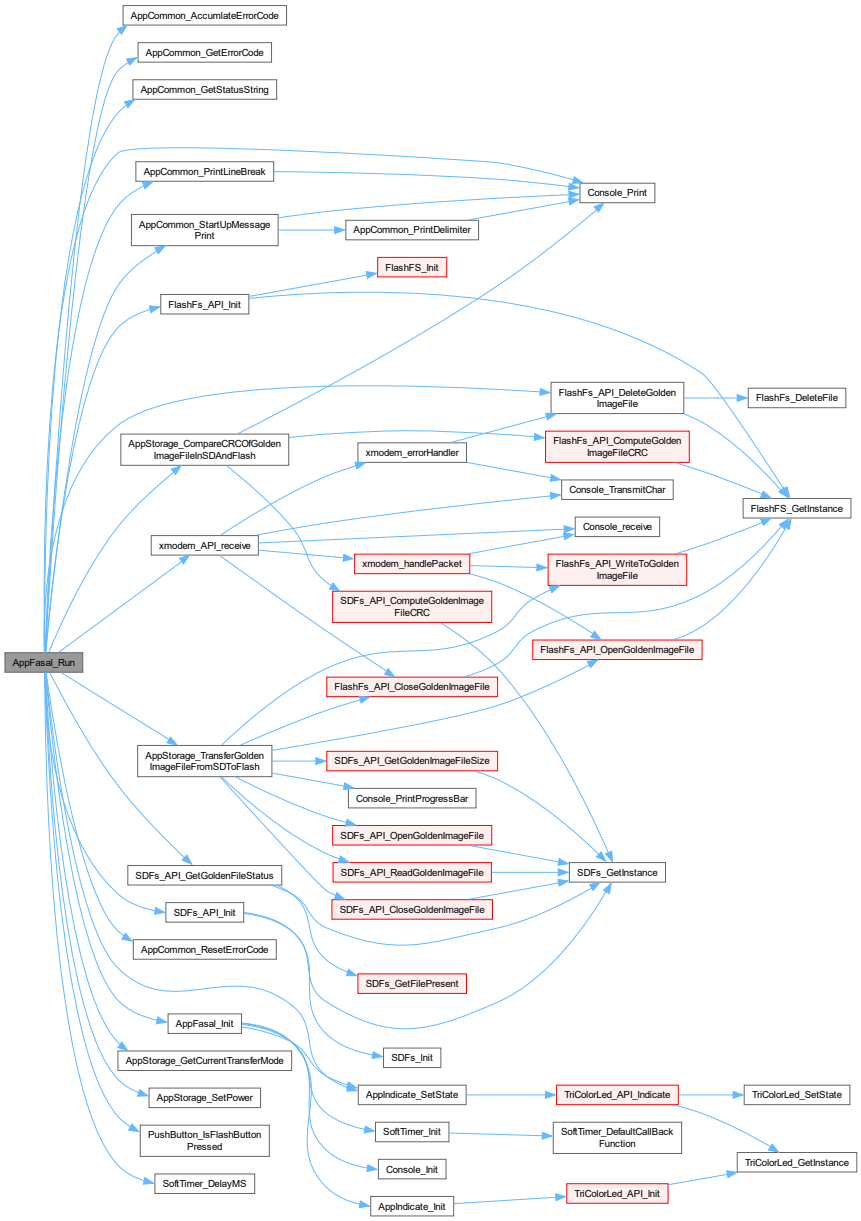
Returns

eAppFasalStates_t

< Set power to External Flash prior to Initializing the same

< Stop powering the external flash since transfer operation is complete

< Errors from previous run if any must be cleared hereHere is the call graph for this function:



5.38 AppFasal.h

Go to the documentation of this file.

```
00001
00013
00014 #ifndef APPFASAL_APPFASAL_H_
00015 #define APPFASAL_APPFASAL_H_
00016
00018
00019 #include "AppCommon.h"
00020
00022
00023
00028 typedef enum
00029 {
```

```
00030     eFASAL_APP_INIT,
00031     eFASAL_APP_STARTUP_MSG,
00032     eFASAL_APP_BUTTON_WAIT,
00033     eFASAL_APP_SD_INIT,
00034     eFASAL_APP_SD_CHECK,
00035     eFASAL_APP_FLASH_INIT,
00036     eFASAL_APP_MODE_SELECTION,
00037     eFASAL_APP_SD_FLASH_TRANSFER,
00038     eFASAL_APP_XMODEM_TRANSFER,
00039     eFASAL_APP_CRC_COMPARE,
00040     eFASAL_APP_TRANSFER_SUCCESS,
00041
00042     eFASAL_APP_SD_FAIL,
00043     eFASAL_APP_SD_FILE_FAIL,
00044     eFASAL_APP_FLASH_FAIL,
00045     eFASAL_APP_TRANSFER_FAIL,
00046     eFASAL_APP_CRC_FAIL,
00047
00048     eFASAL_APP_END,
00049
00050     eFASAL_MAX_STATE,
00051 }eAppFasalStates_t;
00052
00054
00055 eAppFasalStates_t AppFasal_Run();
00056
00058
00059 #endif /* APPFASAL_APPFASAL_H_ */
```

5.39 AppFlash_API.c File Reference

```
#include <assert.h>
#include <string.h>
#include ¨AppFlash_API.h¨
#include ¨W25Qxx.h¨
#include ¨LittleFS_Wrapper.h¨
#include ¨AppConfiguration.h¨
```

Functions

- static sFlashFS_t * FlashFS_GetInstance ()
- static eStorageFSStatus_t FlashFS_Init (sFlashFS_t *const pMe)
- __attribute__ ((unused))
- static eStorageFSStatus_t FlashFs_OpenFileRaw (sFlashFS_t *const pMe, eStorageFileNamesEnums_t fileEnum, eStorageFSOperationModes_t modeEnum)
- static eStorageFSStatus_t FlashFs_CloseFileRaw (sFlashFS_t *const pMe, eStorageFileNamesEnums_t fileEnum)
- static eStorageFSStatus_t FlashFs_ReadFileRaw (sFlashFS_t *const pMe, eStorageFileNamesEnums_t fileEnum, char *const pOutReadBuf, uint32_t bytesToRead, int32_t *const pOutBytesRead)
- static eStorageFSStatus_t FlashFs_WriteFileRaw (sFlashFS_t *const pMe, eStorageFileNamesEnums_t fileEnum, bool IsAppend, const char *const pInWriteBuf, size_t bufSize)
- static eStorageFSStatus_t FlashFs_DeleteFile (const sFlashFS_t *const pMe, eStorageFileNamesEnums_t fileEnum)
- static eStorageFSStatus_t FlashFs_ComputeFileCRC (sFlashFS_t *const pMe, eStorageFileNamesEnums_t fileEnum, uint8_t *const pInOutRamBuf, uint32_t RamBufSize, uint32_t *const pOutCRC)
- eStorageFSStatus_t FlashFs_API_Init ()
- eStorageFSStatus_t FlashFs_API_OpenGoldenImageFile ()
- eStorageFSStatus_t FlashFs_API_WriteToGoldenImageFile (const char *const pInWriteBuf, size_t bufSize)
- eStorageFSStatus_t FlashFs_API_CloseGoldenImageFile ()
- eStorageFSStatus_t FlashFs_API_DeleteGoldenImageFile ()
- eStorageFSStatus_t FlashFs_API_ComputeGoldenImageFileCRC (uint8_t *const pInOutRamBuf, uint32_t RamBufSize, uint32_t *const pOutCRC)

Variables

- static const int gcOpModeToLittleFSModeConverterTable [eFS_MODE_MAX]
- static const char *const gcFilesNamesTable [eFS_MAX]
- static sFlashFS_t gsFlash = {.IsMounted = false}

5.39.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2024-06-07

Copyright

Copyright (c) 2024

5.39.2 Function Documentation

5.39.2.1 FlashFS_GetInstance()

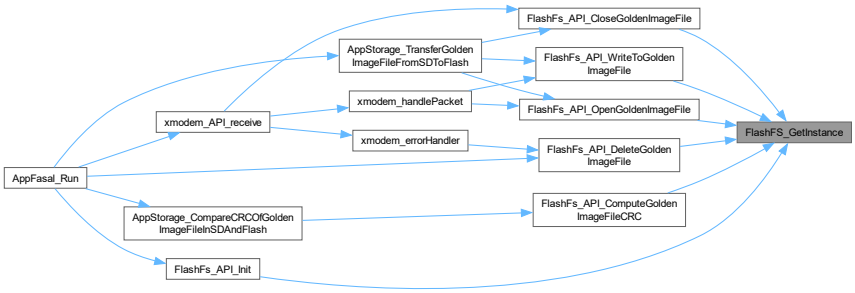
```
static sFlashFS_t * FlashFS_GetInstance ( )  [static]
```

Get FlashFS file instance.

Returns

sFlashFS_t*

Here is the caller graph for this function:



5.39.2.2 FlashFS_Init()

```
static eStorageFSStatus_t FlashFS_Init (
            sFlashFS_t *const pMe )  [static]
```

Initialize lfs.

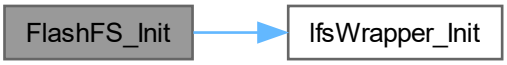
Parameters

pMe	lfs wrapper instance

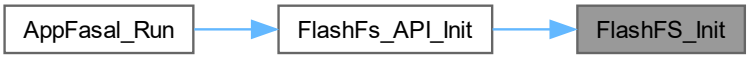
Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.2.3 __attribute__()

```
__attribute__ (
             (unused)  )
```

Deinitialize LFS file system.

Parameters

pMe	lfs wrapper instance

Returns

eStorageFSStatus_t

5.39.2.4 FlashFs_OpenFileRaw()

```
static eStorageFSStatus_t FlashFs_OpenFileRaw (
            sFlashFS_t *const pMe,
```

```
            eStorageFileNamesEnums_t fileEnum,
            eStorageFSOperationModes_t modeEnum )  [static]
```

Open a file in lfs.

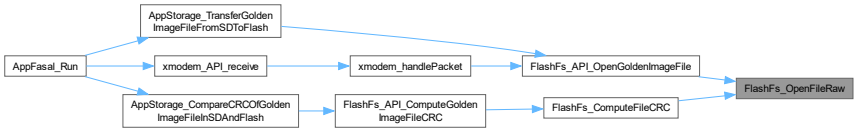
Parameters

pMe	lfs wrapper instance
fileEnum	File to be opened
modeEnum	Mode to open file In

Returns

eStorageFSStatus_t

Here is the caller graph for this function:



5.39.2.5 FlashFs_CloseFileRaw()

```
static eStorageFSStatus_t FlashFs_CloseFileRaw (
            sFlashFS_t *const pMe,
            eStorageFileNamesEnums_t fileEnum )  [static]
```

Close a file in lfs.

Parameters

pMe	lfs wrapper instance
fileEnum	file to be closed

Returns

eStorageFSStatus_t

Here is the caller graph for this function:

5.39.2.6 FlashFs_ReadFileRaw()

```
static eStorageFSStatus_t FlashFs_ReadFileRaw (
            sFlashFS_t *const pMe,
            eStorageFileNamesEnums_t fileEnum,
            char *const pOutReadBuf,
            uint32_t bytesToRead,
            int32_t *const pOutBytesRead )  [static]
```

read from a file in FlashFS

Parameters

pMe	FlashFS wrapper instance
fileEnum	File to be read from
pOutReadBuf	read data will be saved here
bytesToRead	number of bytes to read
pOutBytesRead	number of bytes read is saved here

Returns

eStorageFSStatus_t

Here is the caller graph for this function:

5.39.2.7 FlashFs_WriteFileRaw()

```
static eStorageFSStatus_t FlashFs_WriteFileRaw (
            sFlashFS_t *const pMe,
            eStorageFileNamesEnums_t fileEnum,
            bool IsAppend,
            const char *const pInWriteBuf,
            size_t bufSize )  [static]
```

Write to a file in LFS.

Parameters

pMe	lfs wrapper instance
fileEnum	File to be written into
IsAppend	Unused Kept for API consistency
pInWriteBuf	data to be written is passed here
bufSize	size of buffer passed for writing

Returns

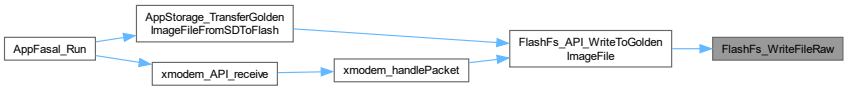
eStorageFSStatus_t

<

Warning

IsAppend Kept for API consistency, File should be opened in Append mode if Appending is needed

Here is the caller graph for this function:



5.39.2.8 FlashFs_DeleteFile()

```
static eStorageFSStatus_t FlashFs_DeleteFile (
            const sFlashFS_t *const pMe,
            eStorageFileNamesEnums_t fileEnum )  [static]
```

Delete File.

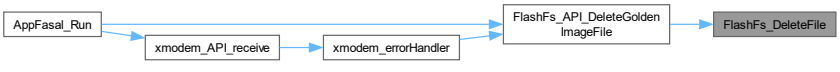
Parameters

pMe	Ifs wrapper instance
fileEnum	File to be deleted

Returns

eStorageFSStatus_t

Here is the caller graph for this function:



5.39.2.9 FlashFs_ComputeFileCRC()

```
static eStorageFSStatus_t FlashFs_ComputeFileCRC (
            sFlashFS_t *const pMe,
```

```
eStorageFileNamesEnums_t fileEnum,
uint8_t *const pInOutRamBuf,
uint32_t RamBufSize,
uint32_t *const pOutCRC )  [static]
```

compute CRC of requested file

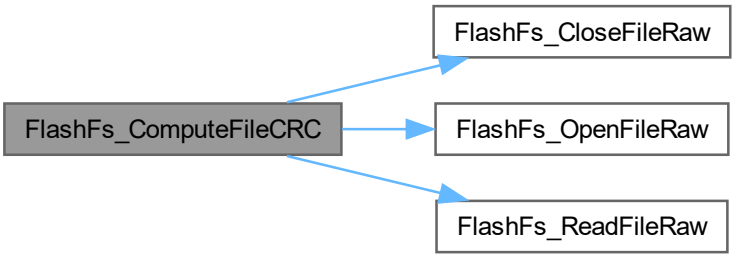
Parameters

pMe	lfs wrapper instance
fileEnum	File whose CRC needs to be computed
pInOutRamBuf	Ram buffer that will be used as temporary storage while computing CRC
RamBufSize	ram buffer size passed
pOutCRC	computed CRC will be saved here

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.39.2.10 FlashFs_API_Init()

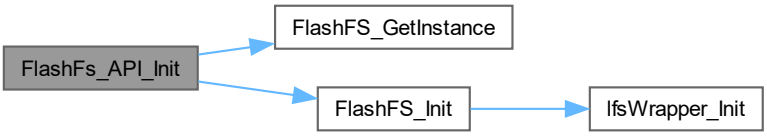
```
eStorageFSStatus_t FlashFs_API_Init ( )
```

API to initialize lfs.

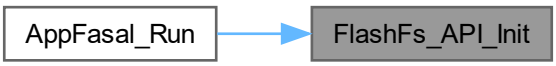
Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.2.11 FlashFs_API_OpenGoldenImageFile()

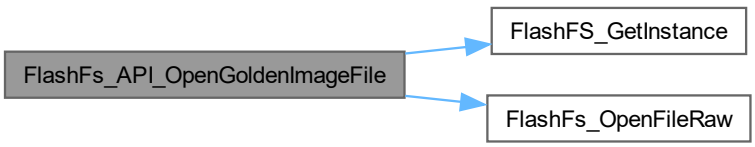
```
eStorageFSStatus_t FlashFs_API_OpenGoldenImageFile ( )
```

Open Golden Image file.

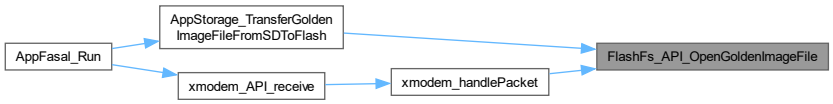
Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.2.12 FlashFs_API_WriteToGoldenImageFile()

```
eStorageFSStatus_t FlashFs_API_WriteToGoldenImageFile (
            const char *const pInWriteBuf,
            size_t bufSize )
```

Write to Golden Image file.

Note

AppStorage_API_OpenGoldenFile must be invoked prior to using this function

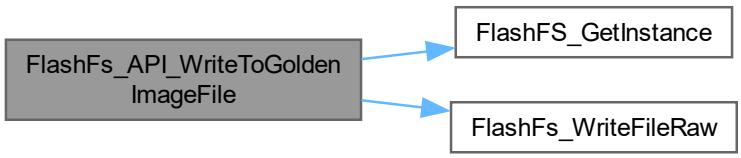
Parameters

pInWriteBuf	Contents to be written to Golden file is passed here
bufSize	write buffer size

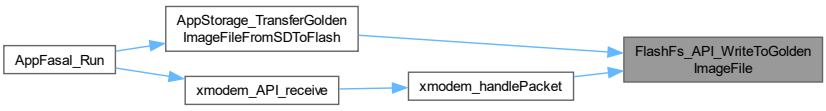
Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.2.13 FlashFs_API_CloseGoldenImageFile()

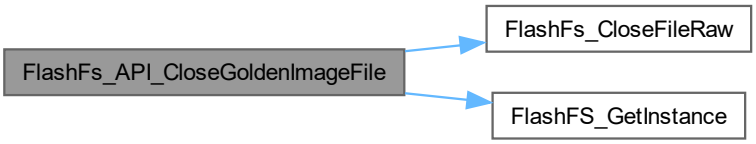
```
eStorageFSStatus_t FlashFs_API_CloseGoldenImageFile ( )
```

Close golden Image file.

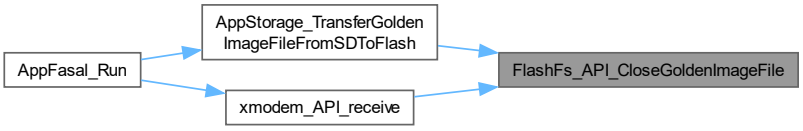
Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.2.14 FlashFs_API_DeleteGoldenImageFile()

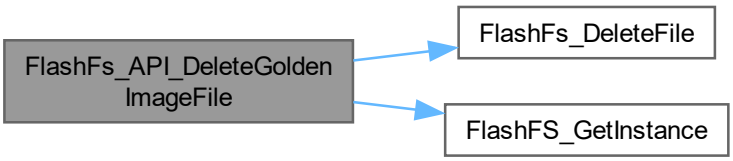
```
eStorageFSStatus_t FlashFs_API_DeleteGoldenImageFile ( )
```

Close golden Image file.

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.39.2.15 FlashFs_API_ComputeGoldenImageFileCRC()

```
eStorageFSStatus_t FlashFs_API_ComputeGoldenImageFileCRC (
            uint8_t *const pInOutRamBuf,
            uint32_t RamBufSize,
            uint32_t *const pOutCRC )
```

compute CRC of golden Image file

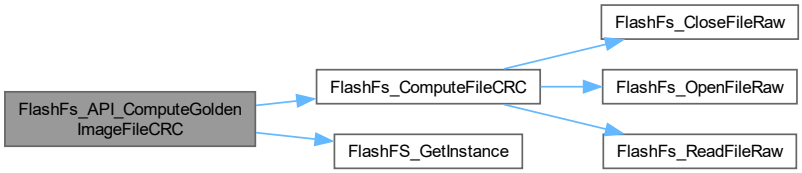
Parameters

pInOutRamBuf	Ram buffer that will be used as temporary storage while computing CRC
RamBufSize	ram buffer size passed
pOutCRC	computed CRC will be saved here

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.39.3 Variable Documentation

5.39.3.1 gcOpModeToLittleFSModeConverterTable

```
const int gcOpModeToLittleFSModeConverterTable[eFS_MODE_MAX]  [static]
```

Initial value:

```
=
{
        [eFS_READONLY]          = (LFS_O_RDONLY),
        [eFS_WRITEONLY]         = (LFS_O_WRONLY),
        [eFS_READ_WRITE]        = (LFS_O_RDWR),
        [eFS_READ_CREATE]       = (LFS_O_RDONLY | LFS_O_CREAT),
        [eFS_WRITE_CREATE]      = (LFS_O_WRONLY | LFS_O_CREAT),
        [eFS_WRITE_APPEND]      = (LFS_O_WRONLY | LFS_O_CREAT | LFS_O_APPEND),
        [eFS_READ_WRITE_CREATE] = (LFS_O_RDWR   | LFS_O_CREAT),
}
```

utility table that converts user file modes to littleFS file modes

5.39.3.2 gcFilesNamesTable

```
const char* const gcFilesNamesTable[eFS_MAX]  [static]
```

Initial value:

```
=
{
        [eFS_GOLDEN_IMAGE]  = "fallback.txt",
        [eFS_FILE_2]        = "file2.txt"
}
```

Map Name enums to File name strings.

5.39.3.3 gsFlash

```
sFlashFS_t gsFlash = {.IsMounted = false}  [static]
```

littleFS wrapper structure instance

5.40 AppFlash_API.h File Reference

```
#include <stdbool.h>
#include "crc.h"
#include "lfs.h"
#include "AppStorageDataStructures.h"
```

Data Structures

- struct sFlashFS_t

Macros

- #define FLASHFS_CRC_INSTANCE (&hcrc)

Functions

- eStorageFSStatus_t FlashFs_API_Init ()
- eStorageFSStatus_t FlashFs_API_OpenGoldenImageFile ()
- eStorageFSStatus_t FlashFs_API_WriteToGoldenImageFile (const char *const pInWriteBuf, size_t bufSize)
- eStorageFSStatus_t FlashFs_API_CloseGoldenImageFile ()
- eStorageFSStatus_t FlashFs_API_DeleteGoldenImageFile ()
- eStorageFSStatus_t FlashFs_API_ComputeGoldenImageFileCRC (uint8_t *const pInOutRamBuf, uint32_t RamBufSize, uint32_t *const pOutCRC)

5.40.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2024-06-07

Copyright

5.40.2 Macro Definition Documentation

5.40.2.1 FLASHFS_CRC_INSTANCE

```
#define FLASHFS_CRC_INSTANCE (&hcrc)
```

CRC instance used by flash module for file integrity check

5.40.3 Function Documentation

5.40.3.1 FlashFs_API_Init()

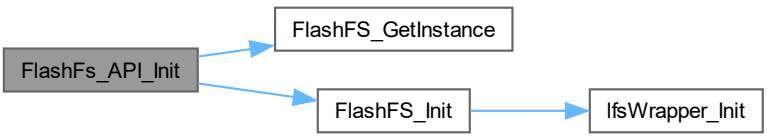
```
eStorageFSStatus_t FlashFs_API_Init ( )
```

API to initialize lfs.

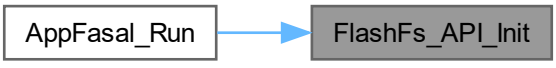
Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:



5.40.3.2 FlashFs_API_OpenGoldenImageFile()

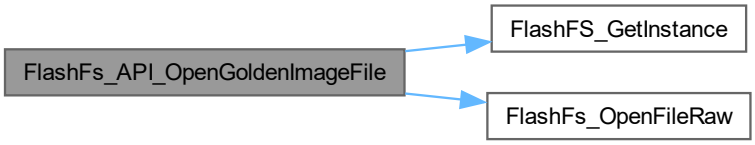
```
eStorageFSStatus_t FlashFs_API_OpenGoldenImageFile ( )
```

Open Golden Image file.

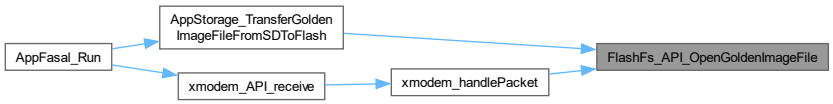
Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:



5.40.3.3 FlashFs_API_WriteToGoldenImageFile()

```
eStorageFSStatus_t FlashFs_API_WriteToGoldenImageFile (
            const char *const pInWriteBuf,
            size_t bufSize )
```

Write to Golden Image file.

Note

AppStorage_API_OpenGoldenFile must be invoked prior to using this function

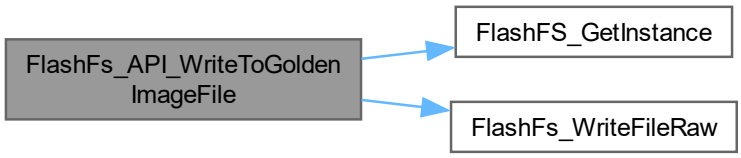
Parameters

pInWriteBuf	Contents to be written to Golden file is passed here
bufSize	write buffer size

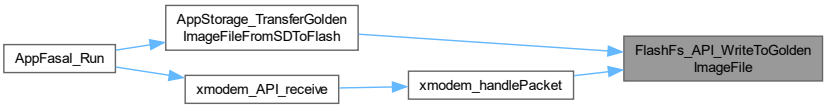
Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:



5.40.3.4 FlashFs_API_CloseGoldenImageFile()

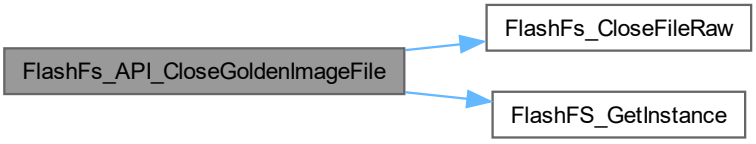
```
eStorageFSStatus_t FlashFs_API_CloseGoldenImageFile ( )
```

Close golden Image file.

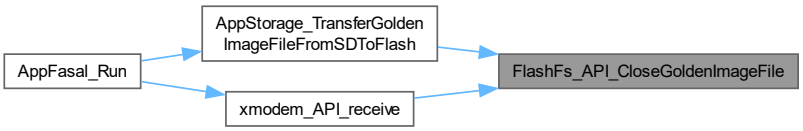
Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:



5.40.3.5 FlashFs_API_DeleteGoldenImageFile()

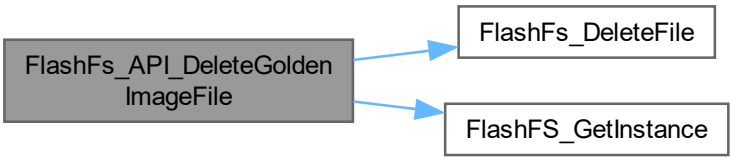
```
eStorageFSStatus_t FlashFs_API_DeleteGoldenImageFile ( )
```

Close golden Image file.

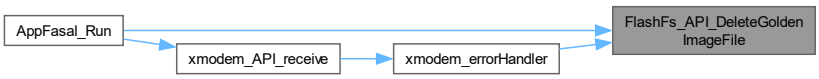
Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:



5.40.3.6 FlashFs_API_ComputeGoldenImageFileCRC()

```
eStorageFSStatus_t FlashFs_API_ComputeGoldenImageFileCRC (
            uint8_t *const pInOutRamBuf,
            uint32_t RamBufSize,
            uint32_t *const pOutCRC )
```

compute CRC of golden Image file

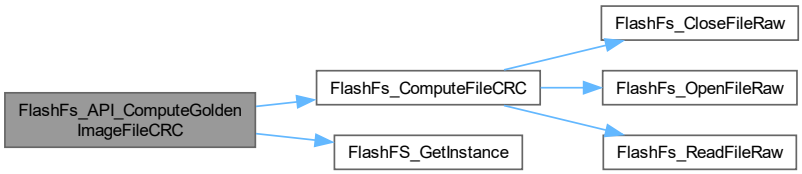
Parameters

pInOutRamBuf	Ram buffer that will be used as temporary storage while computing CRC
RamBufSize	ram buffer size passed
pOutCRC	computed CRC will be saved here

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.41 AppFlash_API.h

Go to the documentation of this file.

```
00001
00013
00014 #ifndef APPSTORAGE_APPFLASHFS_APPFLASH_API_H_
00015 #define APPSTORAGE_APPFLASHFS_APPFLASH_API_H_
00016
00018
00019 #include <stdbool.h>
00020 #include "crc.h"
00021 #include "lfs.h"
00022 #include "AppStorageDataStructures.h"
00023
00025
00026 #define FLASHFS_CRC_INSTANCE    (&hcrc)
00029
00034 typedef struct
00035 {
00036     bool IsMounted;
00037     lfs_t fs;
00038     lfs_file_t fileHandles[eFS_MAX];
00039 }sFlashFS_t;
00040
00042 eStorageFSStatus_t FlashFs_API_Init();
00043 eStorageFSStatus_t FlashFs_API_OpenGoldenImageFile();
00044 eStorageFSStatus_t FlashFs_API_WriteToGoldenImageFile(const char* const pInWriteBuf, size_t bufSize);
```

```
00045 eStorageFSStatus_t FlashFs_API_CloseGoldenImageFile();
00046 eStorageFSStatus_t FlashFs_API_DeleteGoldenImageFile();
00047 eStorageFSStatus_t FlashFs_API_ComputeGoldenImageFileCRC(uint8_t* const pInOutRamBuf, uint32_t
      RamBufSize, uint32_t* const pOutCRC);
00048
00050
00051
00052 #endif /* APPSTORAGE_APPFLASHFS_APPFLASH_API_H_ */
```

5.42 lfs.h

```
00001 /*
00002  * The little filesystem
00003  *
00004  * Copyright (c) 2022, The littlefs authors.
00005  * Copyright (c) 2017, Arm Limited. All rights reserved.
00006  * SPDX-License-Identifier: BSD-3-Clause
00007  */
00008 #ifndef LFS_H
00009 #define LFS_H
00010
00011 #include "lfs_util.h"
00012
00013 #ifdef __cplusplus
00014 extern "C"
00015 {
00016 #endif
00017
00018
00020
00021 // Software library version
00022 // Major (top-nibble), incremented on backwards incompatible changes
00023 // Minor (bottom-nibble), incremented on feature additions
00024 #define LFS_VERSION 0x00020007
00025 #define LFS_VERSION_MAJOR (0xffff & (LFS_VERSION » 16))
00026 #define LFS_VERSION_MINOR (0xffff & (LFS_VERSION »  0))
00027
00028 // Version of On-disk data structures
00029 // Major (top-nibble), incremented on backwards incompatible changes
00030 // Minor (bottom-nibble), incremented on feature additions
00031 #define LFS_DISK_VERSION 0x00020001
00032 #define LFS_DISK_VERSION_MAJOR (0xffff & (LFS_DISK_VERSION » 16))
00033 #define LFS_DISK_VERSION_MINOR (0xffff & (LFS_DISK_VERSION »  0))
00034
00035
00037
00038 // Type definitions
00039 typedef uint32_t lfs_size_t;
00040 typedef uint32_t lfs_off_t;
00041
00042 typedef int32_t  lfs_ssize_t;
00043 typedef int32_t  lfs_soff_t;
00044
00045 typedef uint32_t lfs_block_t;
00046
00047 // Maximum name size in bytes, may be redefined to reduce the size of the
00048 // info struct. Limited to <= 1022. Stored in superblock and must be
00049 // respected by other littlefs drivers.
00050 #ifndef LFS_NAME_MAX
00051 #define LFS_NAME_MAX 255
00052 #endif
00053
00054 // Maximum size of a file in bytes, may be redefined to limit to support other
00055 // drivers. Limited on disk to <= 4294967296. However, above 2147483647 the
00056 // functions lfs_file_seek, lfs_file_size, and lfs_file_tell will return
00057 // incorrect values due to using signed integers. Stored in superblock and
00058 // must be respected by other littlefs drivers.
00059 #ifndef LFS_FILE_MAX
00060 #define LFS_FILE_MAX 2147483647
00061 #endif
00062
00063 // Maximum size of custom attributes in bytes, may be redefined, but there is
00064 // no real benefit to using a smaller LFS_ATTR_MAX. Limited to <= 1022.
00065 #ifndef LFS_ATTR_MAX
00066 #define LFS_ATTR_MAX 1022
00067 #endif
00068
00069 // Possible error codes, these are negative to allow
00070 // valid positive return values
00071 enum lfs_error {
00072     LFS_ERR_OK          = 0,    // No error
00073     LFS_ERR_IO          = -5,   // Error during device operation
00074     LFS_ERR_CORRUPT     = -84,  // Corrupted
```

```
00075     LFS_ERR_NOENT       = -2,   // No directory entry
00076     LFS_ERR_EXIST       = -17,  // Entry already exists
00077     LFS_ERR_NOTDIR      = -20,  // Entry is not a dir
00078     LFS_ERR_ISDIR       = -21,  // Entry is a dir
00079     LFS_ERR_NOTEMPTY    = -39,  // Dir is not empty
00080     LFS_ERR_BADF        = -9,   // Bad file number
00081     LFS_ERR_FBIG        = -27,  // File too large
00082     LFS_ERR_INVAL       = -22,  // Invalid parameter
00083     LFS_ERR_NOSPC       = -28,  // No space left on device
00084     LFS_ERR_NOMEM       = -12,  // No more memory available
00085     LFS_ERR_NOATTR      = -61,  // No data/attr available
00086     LFS_ERR_NAMETOOLONG = -36,  // File name too long
00087 };
00088
00089 // File types
00090 enum lfs_type {
00091     // file types
00092     LFS_TYPE_REG            = 0x001,
00093     LFS_TYPE_DIR            = 0x002,
00094
00095     // internally used types
00096     LFS_TYPE_SPLICE         = 0x400,
00097     LFS_TYPE_NAME           = 0x000,
00098     LFS_TYPE_STRUCT         = 0x200,
00099     LFS_TYPE_USERATTR       = 0x300,
00100     LFS_TYPE_FROM           = 0x100,
00101     LFS_TYPE_TAIL           = 0x600,
00102     LFS_TYPE_GLOBALS        = 0x700,
00103     LFS_TYPE_CRC            = 0x500,
00104
00105     // internally used type specializations
00106     LFS_TYPE_CREATE         = 0x401,
00107     LFS_TYPE_DELETE         = 0x4ff,
00108     LFS_TYPE_SUPERBLOCK     = 0x0ff,
00109     LFS_TYPE_DIRSTRUCT      = 0x200,
00110     LFS_TYPE_CTZSTRUCT      = 0x202,
00111     LFS_TYPE_INLINESTRUCT   = 0x201,
00112     LFS_TYPE_SOFTTAIL       = 0x600,
00113     LFS_TYPE_HARDTAIL       = 0x601,
00114     LFS_TYPE_MOVESTATE      = 0x7ff,
00115     LFS_TYPE_CCRC           = 0x500,
00116     LFS_TYPE_FCRC           = 0x5ff,
00117
00118     // internal chip sources
00119     LFS_FROM_NOOP           = 0x000,
00120     LFS_FROM_MOVE           = 0x101,
00121     LFS_FROM_USERATTRS      = 0x102,
00122 };
00123
00124 // File open flags
00125 enum lfs_open_flags {
00126     // open flags
00127     LFS_O_RDONLY = 1,         // Open a file as read only
00128 #ifndef LFS_READONLY
00129     LFS_O_WRONLY = 2,         // Open a file as write only
00130     LFS_O_RDWR   = 3,         // Open a file as read and write
00131     LFS_O_CREAT  = 0x0100,    // Create a file if it does not exist
00132     LFS_O_EXCL   = 0x0200,    // Fail if a file already exists
00133     LFS_O_TRUNC  = 0x0400,    // Truncate the existing file to zero size
00134     LFS_O_APPEND = 0x0800,    // Move to end of file on every write
00135 #endif
00136
00137     // internally used flags
00138 #ifndef LFS_READONLY
00139     LFS_F_DIRTY   = 0x010000, // File does not match storage
00140     LFS_F_WRITING = 0x020000, // File has been written since last flush
00141 #endif
00142     LFS_F_READING = 0x040000, // File has been read since last flush
00143 #ifndef LFS_READONLY
00144     LFS_F_ERRED   = 0x080000, // An error occurred during write
00145 #endif
00146     LFS_F_INLINE  = 0x100000, // Currently inlined in directory entry
00147 };
00148
00149 // File seek flags
00150 enum lfs_whence_flags {
00151     LFS_SEEK_SET = 0,   // Seek relative to an absolute position
00152     LFS_SEEK_CUR = 1,   // Seek relative to the current file position
00153     LFS_SEEK_END = 2,   // Seek relative to the end of the file
00154 };
00155
00156
00157 // Configuration provided during initialization of the littlefs
00158 struct lfs_config {
00159     // Opaque user provided context that can be used to pass
00160     // information to the block device operations
00161     void *context;
```

```
00162
00163     // Read a region in a block. Negative error codes are propagated
00164     // to the user.
00165     int (*read)(const struct lfs_config *c, lfs_block_t block,
00166             lfs_off_t off, void *buffer, lfs_size_t size);
00167
00168     // Program a region in a block. The block must have previously
00169     // been erased. Negative error codes are propagated to the user.
00170     // May return LFS_ERR_CORRUPT if the block should be considered bad.
00171     int (*prog)(const struct lfs_config *c, lfs_block_t block,
00172             lfs_off_t off, const void *buffer, lfs_size_t size);
00173
00174     // Erase a block. A block must be erased before being programmed.
00175     // The state of an erased block is undefined. Negative error codes
00176     // are propagated to the user.
00177     // May return LFS_ERR_CORRUPT if the block should be considered bad.
00178     int (*erase)(const struct lfs_config *c, lfs_block_t block);
00179
00180     // Sync the state of the underlying block device. Negative error codes
00181     // are propagated to the user.
00182     int (*sync)(const struct lfs_config *c);
00183
00184 #ifdef LFS_THREADSAFE
00185     // Lock the underlying block device. Negative error codes
00186     // are propagated to the user.
00187     int (*lock)(const struct lfs_config *c);
00188
00189     // Unlock the underlying block device. Negative error codes
00190     // are propagated to the user.
00191     int (*unlock)(const struct lfs_config *c);
00192 #endif
00193
00194     // Minimum size of a block read in bytes. All read operations will be a
00195     // multiple of this value.
00196     lfs_size_t read_size;
00197
00198     // Minimum size of a block program in bytes. All program operations will be
00199     // a multiple of this value.
00200     lfs_size_t prog_size;
00201
00202     // Size of an erasable block in bytes. This does not impact ram consumption
00203     // and may be larger than the physical erase size. However, non-inlined
00204     // files take up at minimum one block. Must be a multiple of the read and
00205     // program sizes.
00206     lfs_size_t block_size;
00207
00208     // Number of erasable blocks on the device.
00209     lfs_size_t block_count;
00210
00211     // Number of erase cycles before littlefs evicts metadata logs and moves
00212     // the metadata to another block. Suggested values are in the
00213     // range 100-1000, with large values having better performance at the cost
00214     // of less consistent wear distribution.
00215     //
00216     // Set to -1 to disable block-level wear-leveling.
00217     int32_t block_cycles;
00218
00219     // Size of block caches in bytes. Each cache buffers a portion of a block in
00220     // RAM. The littlefs needs a read cache, a program cache, and one additional
00221     // cache per file. Larger caches can improve performance by storing more
00222     // data and reducing the number of disk accesses. Must be a multiple of the
00223     // read and program sizes, and a factor of the block size.
00224     lfs_size_t cache_size;
00225
00226     // Size of the lookahead buffer in bytes. A larger lookahead buffer
00227     // increases the number of blocks found during an allocation pass. The
00228     // lookahead buffer is stored as a compact bitmap, so each byte of RAM
00229     // can track 8 blocks. Must be a multiple of 8.
00230     lfs_size_t lookahead_size;
00231
00232     // Optional statically allocated read buffer. Must be cache_size.
00233     // By default lfs_malloc is used to allocate this buffer.
00234     void *read_buffer;
00235
00236     // Optional statically allocated program buffer. Must be cache_size.
00237     // By default lfs_malloc is used to allocate this buffer.
00238     void *prog_buffer;
00239
00240     // Optional statically allocated lookahead buffer. Must be lookahead_size
00241     // and aligned to a 32-bit boundary. By default lfs_malloc is used to
00242     // allocate this buffer.
00243     void *lookahead_buffer;
00244
00245     // Optional upper limit on length of file names in bytes. No downside for
00246     // larger names except the size of the info struct which is controlled by
00247     // the LFS_NAME_MAX define. Defaults to LFS_NAME_MAX when zero. Stored in
00248     // superblock and must be respected by other littlefs drivers.
```

```
00249     lfs_size_t name_max;
00250
00251     // Optional upper limit on files in bytes. No downside for larger files
00252     // but must be <= LFS_FILE_MAX. Defaults to LFS_FILE_MAX when zero. Stored
00253     // in superblock and must be respected by other littlefs drivers.
00254     lfs_size_t file_max;
00255
00256     // Optional upper limit on custom attributes in bytes. No downside for
00257     // larger attributes size but must be <= LFS_ATTR_MAX. Defaults to
00258     // LFS_ATTR_MAX when zero.
00259     lfs_size_t attr_max;
00260
00261     // Optional upper limit on total space given to metadata pairs in bytes. On
00262     // devices with large blocks (e.g. 128kB) setting this to a low size (2-8kB)
00263     // can help bound the metadata compaction time. Must be <= block_size.
00264     // Defaults to block_size when zero.
00265     lfs_size_t metadata_max;
00266
00267 #ifdef LFS_MULTIVERSION
00268     // On-disk version to use when writing in the form of 16-bit major version
00269     // + 16-bit minor version. This limiting metadata to what is supported by
00270     // older minor versions. Note that some features will be lost. Defaults to
00271     // to the most recent minor version when zero.
00272     uint32_t disk_version;
00273 #endif
00274 };
00275
00276 // File info structure
00277 struct lfs_info {
00278     // Type of the file, either LFS_TYPE_REG or LFS_TYPE_DIR
00279     uint8_t type;
00280
00281     // Size of the file, only valid for REG files. Limited to 32-bits.
00282     lfs_size_t size;
00283
00284     // Name of the file stored as a null-terminated string. Limited to
00285     // LFS_NAME_MAX+1, which can be changed by redefining LFS_NAME_MAX to
00286     // reduce RAM. LFS_NAME_MAX is stored in superblock and must be
00287     // respected by other littlefs drivers.
00288     char name[LFS_NAME_MAX+1];
00289 };
00290
00291 // Filesystem info structure
00292 struct lfs_fsinfo {
00293     // On-disk version.
00294     uint32_t disk_version;
00295
00296     // Upper limit on the length of file names in bytes.
00297     lfs_size_t name_max;
00298
00299     // Upper limit on the size of files in bytes.
00300     lfs_size_t file_max;
00301
00302     // Upper limit on the size of custom attributes in bytes.
00303     lfs_size_t attr_max;
00304 };
00305
00306 // Custom attribute structure, used to describe custom attributes
00307 // committed atomically during file writes.
00308 struct lfs_attr {
00309     // 8-bit type of attribute, provided by user and used to
00310     // identify the attribute
00311     uint8_t type;
00312
00313     // Pointer to buffer containing the attribute
00314     void *buffer;
00315
00316     // Size of attribute in bytes, limited to LFS_ATTR_MAX
00317     lfs_size_t size;
00318 };
00319
00320 // Optional configuration provided during lfs_file_opencfg
00321 struct lfs_file_config {
00322     // Optional statically allocated file buffer. Must be cache_size.
00323     // By default lfs_malloc is used to allocate this buffer.
00324     void *buffer;
00325
00326     // Optional list of custom attributes related to the file. If the file
00327     // is opened with read access, these attributes will be read from disk
00328     // during the open call. If the file is opened with write access, the
00329     // attributes will be written to disk every file sync or close. This
00330     // write occurs atomically with update to the file's contents.
00331     //
00332     // Custom attributes are uniquely identified by an 8-bit type and limited
00333     // to LFS_ATTR_MAX bytes. When read, if the stored attribute is smaller
00334     // than the buffer, it will be padded with zeros. If the stored attribute
00335     // is larger, then it will be silently truncated. If the attribute is not
```

```
00336     // found, it will be created implicitly.
00337     struct lfs_attr *attrs;
00338
00339     // Number of custom attributes in the list
00340     lfs_size_t attr_count;
00341 };
00342
00343
00345 typedef struct lfs_cache {
00346     lfs_block_t block;
00347     lfs_off_t off;
00348     lfs_size_t size;
00349     uint8_t *buffer;
00350 } lfs_cache_t;
00351
00352 typedef struct lfs_mdir {
00353     lfs_block_t pair[2];
00354     uint32_t rev;
00355     lfs_off_t off;
00356     uint32_t etag;
00357     uint16_t count;
00358     bool erased;
00359     bool split;
00360     lfs_block_t tail[2];
00361 } lfs_mdir_t;
00362
00363 // littlefs directory type
00364 typedef struct lfs_dir {
00365     struct lfs_dir *next;
00366     uint16_t id;
00367     uint8_t type;
00368     lfs_mdir_t m;
00369
00370     lfs_off_t pos;
00371     lfs_block_t head[2];
00372 } lfs_dir_t;
00373
00374 // littlefs file type
00375 typedef struct lfs_file {
00376     struct lfs_file *next;
00377     uint16_t id;
00378     uint8_t type;
00379     lfs_mdir_t m;
00380
00381     struct lfs_ctz {
00382         lfs_block_t head;
00383         lfs_size_t size;
00384     } ctz;
00385
00386     uint32_t flags;
00387     lfs_off_t pos;
00388     lfs_block_t block;
00389     lfs_off_t off;
00390     lfs_cache_t cache;
00391
00392     const struct lfs_file_config *cfg;
00393 } lfs_file_t;
00394
00395 typedef struct lfs_superblock {
00396     uint32_t version;
00397     lfs_size_t block_size;
00398     lfs_size_t block_count;
00399     lfs_size_t name_max;
00400     lfs_size_t file_max;
00401     lfs_size_t attr_max;
00402 } lfs_superblock_t;
00403
00404 typedef struct lfs_gstate {
00405     uint32_t tag;
00406     lfs_block_t pair[2];
00407 } lfs_gstate_t;
00408
00409 // The littlefs filesystem type
00410 typedef struct lfs {
00411     lfs_cache_t rcache;
00412     lfs_cache_t pcache;
00413
00414     lfs_block_t root[2];
00415     struct lfs_mlist {
00416         struct lfs_mlist *next;
00417         uint16_t id;
00418         uint8_t type;
00419         lfs_mdir_t m;
00420     } *mlist;
00421     uint32_t seed;
00422
00423     lfs_gstate_t gstate;
```

```
00424     lfs_gstate_t gdisk;
00425     lfs_gstate_t gdelta;
00426
00427     struct lfs_free {
00428         lfs_block_t off;
00429         lfs_block_t size;
00430         lfs_block_t i;
00431         lfs_block_t ack;
00432         uint32_t *buffer;
00433     } free;
00434
00435     const struct lfs_config *cfg;
00436     lfs_size_t name_max;
00437     lfs_size_t file_max;
00438     lfs_size_t attr_max;
00439
00440 #ifdef LFS_MIGRATE
00441     struct lfs1 *lfs1;
00442 #endif
00443 } lfs_t;
00444
00445
00447
00448 #ifndef LFS_READONLY
00449 // Format a block device with the littlefs
00450 //
00451 // Requires a littlefs object and config struct. This clobbers the littlefs
00452 // object, and does not leave the filesystem mounted. The config struct must
00453 // be zeroed for defaults and backwards compatibility.
00454 //
00455 // Returns a negative error code on failure.
00456 int lfs_format(lfs_t *lfs, const struct lfs_config *config);
00457 #endif
00458
00459 // Mounts a littlefs
00460 //
00461 // Requires a littlefs object and config struct. Multiple filesystems
00462 // may be mounted simultaneously with multiple littlefs objects. Both
00463 // lfs and config must be allocated while mounted. The config struct must
00464 // be zeroed for defaults and backwards compatibility.
00465 //
00466 // Returns a negative error code on failure.
00467 int lfs_mount(lfs_t *lfs, const struct lfs_config *config);
00468
00469 // Unmounts a littlefs
00470 //
00471 // Does nothing besides releasing any allocated resources.
00472 // Returns a negative error code on failure.
00473 int lfs_unmount(lfs_t *lfs);
00474
00476
00477 #ifndef LFS_READONLY
00478 // Removes a file or directory
00479 //
00480 // If removing a directory, the directory must be empty.
00481 // Returns a negative error code on failure.
00482 int lfs_remove(lfs_t *lfs, const char *path);
00483 #endif
00484
00485 #ifndef LFS_READONLY
00486 // Rename or move a file or directory
00487 //
00488 // If the destination exists, it must match the source in type.
00489 // If the destination is a directory, the directory must be empty.
00490 //
00491 // Returns a negative error code on failure.
00492 int lfs_rename(lfs_t *lfs, const char *oldpath, const char *newpath);
00493 #endif
00494
00495 // Find info about a file or directory
00496 //
00497 // Fills out the info structure, based on the specified file or directory.
00498 // Returns a negative error code on failure.
00499 int lfs_stat(lfs_t *lfs, const char *path, struct lfs_info *info);
00500
00501 // Get a custom attribute
00502 //
00503 // Custom attributes are uniquely identified by an 8-bit type and limited
00504 // to LFS_ATTR_MAX bytes. When read, if the stored attribute is smaller than
00505 // the buffer, it will be padded with zeros. If the stored attribute is larger,
00506 // then it will be silently truncated. If no attribute is found, the error
00507 // LFS_ERR_NOATTR is returned and the buffer is filled with zeros.
00508 //
00509 // Returns the size of the attribute, or a negative error code on failure.
00510 // Note, the returned size is the size of the attribute on disk, irrespective
00511 // of the size of the buffer. This can be used to dynamically allocate a buffer
00512 // or check for existence.
```

```
00513 lfs_ssize_t lfs_getattr(lfs_t *lfs, const char *path,
00514         uint8_t type, void *buffer, lfs_size_t size);
00515
00516 #ifndef LFS_READONLY
00517 // Set custom attributes
00518 //
00519 // Custom attributes are uniquely identified by an 8-bit type and limited
00520 // to LFS_ATTR_MAX bytes. If an attribute is not found, it will be
00521 // implicitly created.
00522 //
00523 // Returns a negative error code on failure.
00524 int lfs_setattr(lfs_t *lfs, const char *path,
00525         uint8_t type, const void *buffer, lfs_size_t size);
00526 #endif
00527
00528 #ifndef LFS_READONLY
00529 // Removes a custom attribute
00530 //
00531 // If an attribute is not found, nothing happens.
00532 //
00533 // Returns a negative error code on failure.
00534 int lfs_removeattr(lfs_t *lfs, const char *path, uint8_t type);
00535 #endif
00536
00537
00539
00540 #ifndef LFS_NO_MALLOC
00541 // Open a file
00542 //
00543 // The mode that the file is opened in is determined by the flags, which
00544 // are values from the enum lfs_open_flags that are bitwise-ored together.
00545 //
00546 // Returns a negative error code on failure.
00547 int lfs_file_open(lfs_t *lfs, lfs_file_t *file,
00548         const char *path, int flags);
00549
00550 // if LFS_NO_MALLOC is defined, lfs_file_open() will fail with LFS_ERR_NOMEM
00551 // thus use lfs_file_opencfg() with config.buffer set.
00552 #endif
00553
00554 // Open a file with extra configuration
00555 //
00556 // The mode that the file is opened in is determined by the flags, which
00557 // are values from the enum lfs_open_flags that are bitwise-ored together.
00558 //
00559 // The config struct provides additional config options per file as described
00560 // above. The config struct must remain allocated while the file is open, and
00561 // the config struct must be zeroed for defaults and backwards compatibility.
00562 //
00563 // Returns a negative error code on failure.
00564 int lfs_file_opencfg(lfs_t *lfs, lfs_file_t *file,
00565         const char *path, int flags,
00566         const struct lfs_file_config *config);
00567
00568 // Close a file
00569 //
00570 // Any pending writes are written out to storage as though
00571 // sync had been called and releases any allocated resources.
00572 //
00573 // Returns a negative error code on failure.
00574 int lfs_file_close(lfs_t *lfs, lfs_file_t *file);
00575
00576 // Synchronize a file on storage
00577 //
00578 // Any pending writes are written out to storage.
00579 // Returns a negative error code on failure.
00580 int lfs_file_sync(lfs_t *lfs, lfs_file_t *file);
00581
00582 // Read data from file
00583 //
00584 // Takes a buffer and size indicating where to store the read data.
00585 // Returns the number of bytes read, or a negative error code on failure.
00586 lfs_ssize_t lfs_file_read(lfs_t *lfs, lfs_file_t *file,
00587         void *buffer, lfs_size_t size);
00588
00589 #ifndef LFS_READONLY
00590 // Write data to file
00591 //
00592 // Takes a buffer and size indicating the data to write. The file will not
00593 // actually be updated on the storage until either sync or close is called.
00594 //
00595 // Returns the number of bytes written, or a negative error code on failure.
00596 lfs_ssize_t lfs_file_write(lfs_t *lfs, lfs_file_t *file,
00597         const void *buffer, lfs_size_t size);
00598 #endif
00599
00600 // Change the position of the file
```

```
00601 //
00602 // The change in position is determined by the offset and whence flag.
00603 // Returns the new position of the file, or a negative error code on failure.
00604 lfs_soff_t lfs_file_seek(lfs_t *lfs, lfs_file_t *file,
00605         lfs_soff_t off, int whence);
00606
00607 #ifndef LFS_READONLY
00608 // Truncates the size of the file to the specified size
00609 //
00610 // Returns a negative error code on failure.
00611 int lfs_file_truncate(lfs_t *lfs, lfs_file_t *file, lfs_off_t size);
00612 #endif
00613
00614 // Return the position of the file
00615 //
00616 // Equivalent to lfs_file_seek(lfs, file, 0, LFS_SEEK_CUR)
00617 // Returns the position of the file, or a negative error code on failure.
00618 lfs_soff_t lfs_file_tell(lfs_t *lfs, lfs_file_t *file);
00619
00620 // Change the position of the file to the beginning of the file
00621 //
00622 // Equivalent to lfs_file_seek(lfs, file, 0, LFS_SEEK_SET)
00623 // Returns a negative error code on failure.
00624 int lfs_file_rewind(lfs_t *lfs, lfs_file_t *file);
00625
00626 // Return the size of the file
00627 //
00628 // Similar to lfs_file_seek(lfs, file, 0, LFS_SEEK_END)
00629 // Returns the size of the file, or a negative error code on failure.
00630 lfs_soff_t lfs_file_size(lfs_t *lfs, lfs_file_t *file);
00631
00632
00634
00635 #ifndef LFS_READONLY
00636 // Create a directory
00637 //
00638 // Returns a negative error code on failure.
00639 int lfs_mkdir(lfs_t *lfs, const char *path);
00640 #endif
00641
00642 // Open a directory
00643 //
00644 // Once open a directory can be used with read to iterate over files.
00645 // Returns a negative error code on failure.
00646 int lfs_dir_open(lfs_t *lfs, lfs_dir_t *dir, const char *path);
00647
00648 // Close a directory
00649 //
00650 // Releases any allocated resources.
00651 // Returns a negative error code on failure.
00652 int lfs_dir_close(lfs_t *lfs, lfs_dir_t *dir);
00653
00654 // Read an entry in the directory
00655 //
00656 // Fills out the info structure, based on the specified file or directory.
00657 // Returns a positive value on success, 0 at the end of directory,
00658 // or a negative error code on failure.
00659 int lfs_dir_read(lfs_t *lfs, lfs_dir_t *dir, struct lfs_info *info);
00660
00661 // Change the position of the directory
00662 //
00663 // The new off must be a value previous returned from tell and specifies
00664 // an absolute offset in the directory seek.
00665 //
00666 // Returns a negative error code on failure.
00667 int lfs_dir_seek(lfs_t *lfs, lfs_dir_t *dir, lfs_off_t off);
00668
00669 // Return the position of the directory
00670 //
00671 // The returned offset is only meant to be consumed by seek and may not make
00672 // sense, but does indicate the current position in the directory iteration.
00673 //
00674 // Returns the position of the directory, or a negative error code on failure.
00675 lfs_soff_t lfs_dir_tell(lfs_t *lfs, lfs_dir_t *dir);
00676
00677 // Change the position of the directory to the beginning of the directory
00678 //
00679 // Returns a negative error code on failure.
00680 int lfs_dir_rewind(lfs_t *lfs, lfs_dir_t *dir);
00681
00682
00684
00685 // Find on-disk info about the filesystem
00686 //
00687 // Fills out the fsinfo structure based on the filesystem found on-disk.
00688 // Returns a negative error code on failure.
00689 int lfs_fs_stat(lfs_t *lfs, struct lfs_fsinfo *fsinfo);
```

```
00690
00691 // Finds the current size of the filesystem
00692 //
00693 // Note: Result is best effort. If files share COW structures, the returned
00694 // size may be larger than the filesystem actually is.
00695 //
00696 // Returns the number of allocated blocks, or a negative error code on failure.
00697 lfs_ssize_t lfs_fs_size(lfs_t *lfs);
00698
00699 // Traverse through all blocks in use by the filesystem
00700 //
00701 // The provided callback will be called with each block address that is
00702 // currently in use by the filesystem. This can be used to determine which
00703 // blocks are in use or how much of the storage is available.
00704 //
00705 // Returns a negative error code on failure.
00706 int lfs_fs_traverse(lfs_t *lfs, int (*cb)(void*, lfs_block_t), void *data);
00707
00708 #ifndef LFS_READONLY
00709 // Attempt to make the filesystem consistent and ready for writing
00710 //
00711 // Calling this function is not required, consistency will be implicitly
00712 // enforced on the first operation that writes to the filesystem, but this
00713 // function allows the work to be performed earlier and without other
00714 // filesystem changes.
00715 //
00716 // Returns a negative error code on failure.
00717 int lfs_fs_mkconsistent(lfs_t *lfs);
00718 #endif
00719
00720 #ifndef LFS_READONLY
00721 #ifdef LFS_MIGRATE
00722 // Attempts to migrate a previous version of littlefs
00723 //
00724 // Behaves similarly to the lfs_format function. Attempts to mount
00725 // the previous version of littlefs and update the filesystem so it can be
00726 // mounted with the current version of littlefs.
00727 //
00728 // Requires a littlefs object and config struct. This clobbers the littlefs
00729 // object, and does not leave the filesystem mounted. The config struct must
00730 // be zeroed for defaults and backwards compatibility.
00731 //
00732 // Returns a negative error code on failure.
00733 int lfs_migrate(lfs_t *lfs, const struct lfs_config *cfg);
00734 #endif
00735 #endif
00736
00737
00738 #ifdef __cplusplus
00739 } /* extern "C" */
00740 #endif
00741
00742 #endif
```

5.43 lfs_util.h

```
00001 /*
00002  * lfs utility functions
00003  *
00004  * Copyright (c) 2022, The littlefs authors.
00005  * Copyright (c) 2017, Arm Limited. All rights reserved.
00006  * SPDX-License-Identifier: BSD-3-Clause
00007  */
00008 #ifndef LFS_UTIL_H
00009 #define LFS_UTIL_H
00010
00011 // Users can override lfs_util.h with their own configuration by defining
00012 // LFS_CONFIG as a header file to include (-DLFS_CONFIG=lfs_config.h).
00013 //
00014 // If LFS_CONFIG is used, none of the default utils will be emitted and must be
00015 // provided by the config file. To start, I would suggest copying lfs_util.h
00016 // and modifying as needed.
00017 #ifdef LFS_CONFIG
00018 #define LFS_STRINGIZE(x) LFS_STRINGIZE2(x)
00019 #define LFS_STRINGIZE2(x) #x
00020 #include LFS_STRINGIZE(LFS_CONFIG)
00021 #else
00022
00023 // System includes
00024 #include <stdint.h>
00025 #include <stdbool.h>
00026 #include <string.h>
00027 #include <inttypes.h>
```

```
00028
00029 #ifndef LFS_NO_MALLOC
00030 #include <stdlib.h>
00031 #endif
00032 #ifndef LFS_NO_ASSERT
00033 #include <assert.h>
00034 #endif
00035 #if !defined(LFS_NO_DEBUG) || \
00036         !defined(LFS_NO_WARN) || \
00037         !defined(LFS_NO_ERROR) || \
00038         defined(LFS_YES_TRACE)
00039 #include <stdio.h>
00040 #endif
00041
00042 #ifdef __cplusplus
00043 extern "C"
00044 {
00045 #endif
00046
00047
00048 // Macros, may be replaced by system specific wrappers. Arguments to these
00049 // macros must not have side-effects as the macros can be removed for a smaller
00050 // code footprint
00051 // Logging functions
00052 #ifndef LFS_TRACE
00053 #ifdef LFS_YES_TRACE
00054 #define LFS_TRACE_(fmt, ...) \
00055     printf("%s:%d:trace: " fmt "%s\n", __FILE__, __LINE__, __VA_ARGS__)
00056 #define LFS_TRACE(...) LFS_TRACE_(__VA_ARGS__, "")
00057 #else
00058 #define LFS_TRACE(...)
00059 #endif
00060 #endif
00061
00062 #ifndef LFS_DEBUG
00063 #ifndef LFS_NO_DEBUG
00064 #define LFS_DEBUG_(fmt, ...) \
00065     printf("%s:%d:debug: " fmt "%s\n", __FILE__, __LINE__, __VA_ARGS__)
00066 #define LFS_DEBUG(...) LFS_DEBUG_(__VA_ARGS__, "")
00067 #else
00068 #define LFS_DEBUG(...)
00069 #endif
00070 #endif
00071
00072 #ifndef LFS_WARN
00073 #ifndef LFS_NO_WARN
00074 #define LFS_WARN_(fmt, ...) \
00075     printf("%s:%d:warn: " fmt "%s\n", __FILE__, __LINE__, __VA_ARGS__)
00076 #define LFS_WARN(...) LFS_WARN_(__VA_ARGS__, "")
00077 #else
00078 #define LFS_WARN(...)
00079 #endif
00080 #endif
00081
00082 #ifndef LFS_ERROR
00083 #ifndef LFS_NO_ERROR
00084 #define LFS_ERROR_(fmt, ...) \
00085     printf("%s:%d:error: " fmt "%s\n", __FILE__, __LINE__, __VA_ARGS__)
00086 #define LFS_ERROR(...) LFS_ERROR_(__VA_ARGS__, "")
00087 #else
00088 #define LFS_ERROR(...)
00089 #endif
00090 #endif
00091
00092 // Runtime assertions
00093 #ifndef LFS_ASSERT
00094 #ifndef LFS_NO_ASSERT
00095 #define LFS_ASSERT(test) assert(test)
00096 #else
00097 #define LFS_ASSERT(test)
00098 #endif
00099 #endif
00100
00101
00102 // Builtin functions, these may be replaced by more efficient
00103 // toolchain-specific implementations. LFS_NO_INTRINSICS falls back to a more
00104 // expensive basic C implementation for debugging purposes
00105
00106 // Min/max functions for unsigned 32-bit numbers
00107 static inline uint32_t lfs_max(uint32_t a, uint32_t b) {
00108     return (a > b) ? a : b;
00109 }
00110
00111 static inline uint32_t lfs_min(uint32_t a, uint32_t b) {
00112     return (a < b) ? a : b;
00113 }
00114
```

```
00115 // Align to nearest multiple of a size
00116 static inline uint32_t lfs_aligndown(uint32_t a, uint32_t alignment) {
00117     return a - (a % alignment);
00118 }
00119
00120 static inline uint32_t lfs_alignup(uint32_t a, uint32_t alignment) {
00121     return lfs_aligndown(a + alignment-1, alignment);
00122 }
00123
00124 // Find the smallest power of 2 greater than or equal to a
00125 static inline uint32_t lfs_npw2(uint32_t a) {
00126 #if !defined(LFS_NO_INTRINSICS) && (defined(__GNUC__) || defined(__CC_ARM))
00127     return 32 - __builtin_clz(a-1);
00128 #else
00129     uint32_t r = 0;
00130     uint32_t s;
00131     a -= 1;
00132     s = (a > 0xffff) « 4; a »= s; r |= s;
00133     s = (a > 0xff  ) « 3; a »= s; r |= s;
00134     s = (a > 0xf   ) « 2; a »= s; r |= s;
00135     s = (a > 0x3   ) « 1; a »= s; r |= s;
00136     return (r | (a » 1)) + 1;
00137 #endif
00138 }
00139
00140 // Count the number of trailing binary zeros in a
00141 // lfs_ctz(0) may be undefined
00142 static inline uint32_t lfs_ctz(uint32_t a) {
00143 #if !defined(LFS_NO_INTRINSICS) && defined(__GNUC__)
00144     return __builtin_ctz(a);
00145 #else
00146     return lfs_npw2((a & -a) + 1) - 1;
00147 #endif
00148 }
00149
00150 // Count the number of binary ones in a
00151 static inline uint32_t lfs_popc(uint32_t a) {
00152 #if !defined(LFS_NO_INTRINSICS) && (defined(__GNUC__) || defined(__CC_ARM))
00153     return __builtin_popcount(a);
00154 #else
00155     a = a - ((a » 1) & 0x55555555);
00156     a = (a & 0x33333333) + ((a » 2) & 0x33333333);
00157     return (((a + (a » 4)) & 0xf0f0f0f) * 0x1010101) » 24;
00158 #endif
00159 }
00160
00161 // Find the sequence comparison of a and b, this is the distance
00162 // between a and b ignoring overflow
00163 static inline int lfs_scmp(uint32_t a, uint32_t b) {
00164     return (int)(unsigned)(a - b);
00165 }
00166
00167 // Convert between 32-bit little-endian and native order
00168 static inline uint32_t lfs_fromle32(uint32_t a) {
00169 #if (defined(  BYTE_ORDER  ) && defined(  ORDER_LITTLE_ENDIAN  ) &&   BYTE_ORDER   ==
      ORDER_LITTLE_ENDIAN  ) || \
00170     (defined(__BYTE_ORDER  ) && defined(__ORDER_LITTLE_ENDIAN  ) && __BYTE_ORDER   ==
      __ORDER_LITTLE_ENDIAN  ) || \
00171     (defined(__BYTE_ORDER__) && defined(__ORDER_LITTLE_ENDIAN__) && __BYTE_ORDER__ ==
      __ORDER_LITTLE_ENDIAN__)
00172     return a;
00173 #elif !defined(LFS_NO_INTRINSICS) && ( \
00174     (defined(  BYTE_ORDER  ) && defined(  ORDER_BIG_ENDIAN  ) &&   BYTE_ORDER   ==   ORDER_BIG_ENDIAN
      ) || \
00175     (defined(__BYTE_ORDER  ) && defined(__ORDER_BIG_ENDIAN  ) && __BYTE_ORDER   == __ORDER_BIG_ENDIAN
      ) || \
00176     (defined(__BYTE_ORDER__) && defined(__ORDER_BIG_ENDIAN__) && __BYTE_ORDER__ ==
      __ORDER_BIG_ENDIAN__))
00177     return __builtin_bswap32(a);
00178 #else
00179     return (((uint8_t*)&a)[0] «  0) |
00180            (((uint8_t*)&a)[1] «  8) |
00181            (((uint8_t*)&a)[2] « 16) |
00182            (((uint8_t*)&a)[3] « 24);
00183 #endif
00184 }
00185
00186 static inline uint32_t lfs_tole32(uint32_t a) {
00187     return lfs_fromle32(a);
00188 }
00189
00190 // Convert between 32-bit big-endian and native order
00191 static inline uint32_t lfs_frombe32(uint32_t a) {
00192 #if !defined(LFS_NO_INTRINSICS) && ( \
00193     (defined(  BYTE_ORDER  ) && defined(  ORDER_LITTLE_ENDIAN  ) &&   BYTE_ORDER   ==
      ORDER_LITTLE_ENDIAN  ) || \
00194     (defined(__BYTE_ORDER  ) && defined(__ORDER_LITTLE_ENDIAN  ) && __BYTE_ORDER   ==
```

```
    __ORDER_LITTLE_ENDIAN__ ) || \
00195     (defined(__BYTE_ORDER__) && defined(__ORDER_LITTLE_ENDIAN__) && __BYTE_ORDER__ ==
    __ORDER_LITTLE_ENDIAN__))
00196     return __builtin_bswap32(a);
00197 #elif (defined( BYTE_ORDER ) && defined( ORDER_BIG_ENDIAN ) &&  BYTE_ORDER  ==
    ORDER_BIG_ENDIAN ) || \
00198     (defined(__BYTE_ORDER__) && defined(__ORDER_BIG_ENDIAN__) && __BYTE_ORDER__ == __ORDER_BIG_ENDIAN
    __) || \
00199     (defined(__BYTE_ORDER__) && defined(__ORDER_BIG_ENDIAN__) && __BYTE_ORDER__ ==
    __ORDER_BIG_ENDIAN__)
00200     return a;
00201 #else
00202     return (((uint8_t*)&a)[0] << 24) |
00203            (((uint8_t*)&a)[1] << 16) |
00204            (((uint8_t*)&a)[2] <<  8) |
00205            (((uint8_t*)&a)[3] <<  0);
00206 #endif
00207 }
00208
00209 static inline uint32_t lfs_tobe32(uint32_t a) {
00210     return lfs_frombe32(a);
00211 }
00212
00213 // Calculate CRC-32 with polynomial = 0x04c11db7
00214 uint32_t lfs_crc(uint32_t crc, const void *buffer, size_t size);
00215
00216 // Allocate memory, only used if buffers are not provided to littlefs
00217 // Note, memory must be 64-bit aligned
00218 static inline void *lfs_malloc(size_t size) {
00219 #ifndef LFS_NO_MALLOC
00220     return malloc(size);
00221 #else
00222     (void)size;
00223     return NULL;
00224 #endif
00225 }
00226
00227 // Deallocate memory, only used if buffers are not provided to littlefs
00228 static inline void lfs_free(void *p) {
00229 #ifndef LFS_NO_MALLOC
00230     free(p);
00231 #else
00232     (void)p;
00233 #endif
00234 }
00235
00236
00237 #ifdef __cplusplus
00238 } /* extern "C" */
00239 #endif
00240
00241 #endif
00242 #endif
```

5.44 LittleFS_Wrapper.c File Reference

```
#include ¨LittleFS_Wrapper.h¨
#include ¨W25Qxx.h¨
#include ¨Console.h¨
#include ¨AppConfiguration.h¨
```

Macros

- #define **LFS_READ_SIZE** (128)
- #define **LFS_PROG_SIZE** (LFS_READ_SIZE)
- #define **LFS_LOOKAHEAD_SIZE** (LFS_READ_SIZE / 8)
- #define **LFS_BLOCK_SIZE** (65536)
- #define **LFS_BLOCK_COUNT** (128)
- #define **LFS_BLOCK_CYCLES** (-1)
- #define **LFS_CACHE_SIZE** (64 % LFS_PROG_SIZE == 0 ? 64 : LFS_PROG_SIZE)
- #define **LFS_ERASE_VALUE** (0xff)
- #define **LFS_ERASE_CYCLES** (-1)
- #define **LFS_BADBLOCK_BEHAVIOR** (LFS_TESTBD_BADBLOCK_PROGERROR)

Functions

- static __attribute__ ((aligned(32)))
- static int lfs_device_prog (const struct lfs_config *c, lfs_block_t block, lfs_off_t off, const void *buffer, lfs_size_t size)
- static int lfs_device_erase (const struct lfs_config *c, lfs_block_t block)
- static int lfs_device_sync (const struct lfs_config *c)
- int lfsWrapper_Init (lfs_t *const plfs)

Variables

- static uint8_t **lfs_read_buf** [LFS_READ_SIZE]
- static uint8_t **lfs_prog_buf** [LFS_PROG_SIZE]
- static const struct lfs_config cfg

5.44.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-08-16

Copyright

Copyright (c) 2023

5.44.2 Function Documentation

5.44.2.1 lfs_device_prog()

```
static int lfs_device_prog (
            const struct lfs_config * c,
            lfs_block_t block,
            lfs_off_t off,
            const void * buffer,
            lfs_size_t size )  [static]
```

read function provided to LitleFS

Parameters

c	
block	
off	
buffer	
size	

Returns

int

5.44.2.2 lfs_device_erase()

```
static int lfs_device_erase (
            const struct lfs_config * c,
            lfs_block_t block )  [static]
```

Erase function provided to LitleFS.

Parameters

c	
block	

Returns

int

5.44.2.3 lfs_device_sync()

```
static int lfs_device_sync (
            const struct lfs_config * c )  [static]
```

Erase function provided to LitleFS. Not supported by W25qxx.

Parameters

c	

Returns

int

5.44.2.4 lfsWrapper_Init()

```
int lfsWrapper_Init (
            lfs_t *const plfs )
```

Wrapper around littleFS initialization.

Parameters

plfs	pointer to lfs structure

Returns

int non zero if error

< reformat if we can't mount the filesystem

< this should only happen on the first bootHere is the caller graph for this function:

5.44.3 Variable Documentation

5.44.3.1 cfg

```
const struct lfs_config cfg  [static]
```

Initial value:

```
= {

    .read  = lfs_device_read,
    .prog  = lfs_device_prog,
    .erase = lfs_device_erase,
    .sync  = lfs_device_sync,


    .read_size = LFS_READ_SIZE,
    .prog_size = LFS_PROG_SIZE,
    .block_size = LFS_BLOCK_SIZE,
    .block_count = LFS_BLOCK_COUNT,
    .cache_size = LFS_CACHE_SIZE,
    .lookahead_size = LFS_LOOKAHEAD_SIZE,
    .block_cycles = LFS_BLOCK_CYCLES,

    .read_buffer = lfs_read_buf,
    .prog_buffer = lfs_prog_buf,
    .lookahead_buffer = lfs_lookahead_buf,
}
```

configuration of the filesystem is provided by this struct

5.45 LittleFS_Wrapper.h File Reference

```
#include <stdint.h>
#include <stdbool.h>
#include ¨lfs.h¨
```

Functions

- int lfsWrapper_Init (lfs_t *const plfs)
- bool **lfs_Test** (void)

5.45.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-08-16

Copyright

Copyright (c) 2023

5.45.2 Function Documentation

5.45.2.1 lfsWrapper_Init()

```
int lfsWrapper_Init (
            lfs_t *const plfs )
```

Wrapper around littleFS initialization.

Parameters

plfs	pointer to lfs structure

Returns

int non zero if error

< reformat if we can't mount the filesystem

< this should only happen on the first bootHere is the caller graph for this function:

5.46 LittleFS_Wrapper.h

Go to the documentation of this file.

```
00001
00013
00014 #ifndef _LITTLE_FS_WRAPPER_H    /* Guard against multiple inclusion */
00015 #define _LITTLE_FS_WRAPPER_H
00016
00018
00019 #include <stdint.h>
00020 #include <stdbool.h>
00021 #include "lfs.h"
00022
00024
00025 int lfsWrapper_Init(lfs_t* const plfs);
00026 bool lfs_Test(void);
00027
00029
00030
00031
00032 #endif /* _LITTLE_FS_WRAPPER_H */
00033
```

5.47 W25Qxx.c File Reference

```
#include <stddef.h>
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include ¨W25qxx.h¨
#include ¨DebugPrint.h¨
#include ¨AppConfiguration.h¨
```

Macros

- #define **W25QXX_DUMMY_BYTE** (0xA5)
- #define **W25qxx_Delay**(delay) HAL_Delay(delay)

Functions

- void **FLASH_SS_Clear** ()
- void **FLASH_SS_Set** ()
- uint8_t **W25qxx_Spi** (uint8_t Data)
- uint32_t **W25qxx_ReadID** (void)
- void **W25qxx_ReadUniqID** (void)
- void **W25qxx_WriteEnable** (void)
- void **W25qxx_WriteDisable** (void)
- uint8_t **W25qxx_ReadStatusRegister** (uint8_t SelectStatusRegister_1_2_3)
- void **W25qxx_WriteStatusRegister** (uint8_t SelectStatusRegister_1_2_3, uint8_t Data)
- void **W25qxx_WaitForWriteEnd** (void)
- eW25qxxStatus **W25qxx_Init** (void)
- eW25qxxStatus **W25qxx_EraseChip** (void)
- eW25qxxStatus **W25qxx_EraseSector** (uint32_t SectorAddr)
- eW25qxxStatus **W25qxx_EraseBlock** (uint32_t BlockAddr)
- uint32_t **W25qxx_PageToSector** (uint32_t PageAddress)
- uint32_t **W25qxx_PageToBlock** (uint32_t PageAddress)
- uint32_t **W25qxx_SectorToBlock** (uint32_t SectorAddress)
- uint32_t **W25qxx_SectorToPage** (uint32_t SectorAddress)
- uint32_t **W25qxx_BlockToPage** (uint32_t BlockAddress)

- bool **W25qxx_IsEmptyPage** (uint32_t Page_Address, uint32_t OffsetInByte, uint32_t NumByteToCheck_↩ up_to_PageSize)
- bool **W25qxx_IsEmptySector** (uint32_t Sector_Address, uint32_t OffsetInByte, uint32_t NumByteTo↩ Check_up_to_SectorSize)
- bool **W25qxx_IsEmptyBlock** (uint32_t Block_Address, uint32_t OffsetInByte, uint32_t NumByteToCheck↩ _up_to_BlockSize)
- eW25qxxStatus **W25qxx_WriteByte** (uint8_t pBuffer, uint32_t WriteAddr_inBytes)
- eW25qxxStatus **W25qxx_WritePage** (uint8_t ∗pBuffer, uint32_t Page_Address, uint32_t OffsetInByte, uint32_t NumByteToWrite_up_to_PageSize)
- eW25qxxStatus **W25qxx_WriteSector** (uint8_t ∗pBuffer, uint32_t Sector_Address, uint32_t OffsetInByte, uint32_t NumByteToWrite_up_to_SectorSize)
- eW25qxxStatus **W25qxx_WriteBlock** (uint8_t ∗pBuffer, uint32_t Block_Address, uint32_t OffsetInByte, uint32_t NumByteToWrite_up_to_BlockSize)
- eW25qxxStatus **W25qxx_ReadByte** (uint8_t ∗pBuffer, uint32_t Bytes_Address)
- eW25qxxStatus **W25qxx_ReadBytes** (uint8_t ∗pBuffer, uint32_t ReadAddr, uint32_t NumByteToRead)
- eW25qxxStatus **W25qxx_ReadPage** (uint8_t ∗pBuffer, uint32_t Page_Address, uint32_t OffsetInByte, uint32_t NumByteToRead_up_to_PageSize)
- eW25qxxStatus **W25qxx_ReadSector** (uint8_t ∗pBuffer, uint32_t Sector_Address, uint32_t OffsetInByte, uint32_t NumByteToRead_up_to_SectorSize)
- eW25qxxStatus **W25qxx_ReadBlock** (uint8_t ∗pBuffer, uint32_t Block_Address, uint32_t OffsetInByte, uint32_t NumByteToRead_up_to_BlockSize)

Variables

- static w25qxx_t gW25qxxDev

5.47.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-08-05

Copyright

5.47.2 Variable Documentation

5.47.2.1 gW25qxxDev

```
w25qxx_t gW25qxxDev  [static]
```

Global Device instance

5.48 W25Qxx.h File Reference

```
#include <stdint.h>
#include <stdbool.h>
#include ¨spi.h¨
#include ¨gpio.h¨
```

Data Structures

- struct df_dataparam
- struct w25qxx_t

Macros

- #define **W25QXXH_SPI_HANDLE** (&hspi2)
- #define **W25QXXH_SPI_TIMEOUT_MS** (100)
- #define **W25QXXH_SPI_CS_PORT** (SPI2_NSS_GPIO_Port)
- #define **W25XXH_SPI_CS_PIN** (SPI2_NSS_Pin)
- #define **_W25QXX_DEBUG** (0)
- #define **DF_MAX_NO_PAGE** 65536
- #define **DF_PAGE_SIZE** 256
- #define **DF_SECTOR_SIZE** 16
- #define **DF_SBLOCK_SIZE** 128
- #define **DF_BBLOCK_SIZE** 256
- #define **DF_MAX_SECTOR** 4096
- #define **DF_MAX_SBLOCK** 512
- #define **DF_MAX_BBLOCK** 256

Typedefs

- typedef int32_t **eW25qxxStatus**

Enumerations

- enum **W25QXX_ID_t** {
  W25Q10 =1 , **W25Q20** , **W25Q40** , **W25Q80** ,
  W25Q16 , **W25Q32** , **W25Q64** , **W25Q128** ,
  W25Q256 , **W25Q512** }

Functions

- eW25qxxStatus **W25qxx_Init** (void)
- eW25qxxStatus **W25qxx_EraseChip** (void)
- eW25qxxStatus **W25qxx_EraseSector** (uint32_t SectorAddr)
- eW25qxxStatus **W25qxx_EraseBlock** (uint32_t BlockAddr)
- uint32_t **W25qxx_PageToSector** (uint32_t PageAddress)
- uint32_t **W25qxx_PageToBlock** (uint32_t PageAddress)
- uint32_t **W25qxx_SectorToBlock** (uint32_t SectorAddress)
- uint32_t **W25qxx_SectorToPage** (uint32_t SectorAddress)
- uint32_t **W25qxx_BlockToPage** (uint32_t BlockAddress)
- bool **W25qxx_IsEmptyPage** (uint32_t Page_Address, uint32_t OffsetInByte, uint32_t NumByteToCheck_↩ up_to_PageSize)
- bool **W25qxx_IsEmptySector** (uint32_t Sector_Address, uint32_t OffsetInByte, uint32_t NumByteTo↩ Check_up_to_SectorSize)
- bool **W25qxx_IsEmptyBlock** (uint32_t Block_Address, uint32_t OffsetInByte, uint32_t NumByteToCheck↩ _up_to_BlockSize)
- eW25qxxStatus **W25qxx_WriteByte** (uint8_t pBuffer, uint32_t WriteAddr_inBytes)
- eW25qxxStatus **W25qxx_WritePage** (uint8_t ∗pBuffer, uint32_t Page_Address, uint32_t OffsetInByte, uint32_t NumByteToWrite_up_to_PageSize)
- eW25qxxStatus **W25qxx_WriteSector** (uint8_t ∗pBuffer, uint32_t Sector_Address, uint32_t OffsetInByte, uint32_t NumByteToWrite_up_to_SectorSize)
- eW25qxxStatus **W25qxx_WriteBlock** (uint8_t ∗pBuffer, uint32_t Block_Address, uint32_t OffsetInByte, uint32_t NumByteToWrite_up_to_BlockSize)
- eW25qxxStatus **W25qxx_ReadByte** (uint8_t ∗pBuffer, uint32_t Bytes_Address)
- eW25qxxStatus **W25qxx_ReadBytes** (uint8_t ∗pBuffer, uint32_t ReadAddr, uint32_t NumByteToRead)
- eW25qxxStatus **W25qxx_ReadPage** (uint8_t ∗pBuffer, uint32_t Page_Address, uint32_t OffsetInByte, uint32_t NumByteToRead_up_to_PageSize)
- eW25qxxStatus **W25qxx_ReadSector** (uint8_t ∗pBuffer, uint32_t Sector_Address, uint32_t OffsetInByte, uint32_t NumByteToRead_up_to_SectorSize)
- eW25qxxStatus **W25qxx_ReadBlock** (uint8_t ∗pBuffer, uint32_t Block_Address, uint32_t OffsetInByte, uint32_t NumByteToRead_up_to_BlockSize)
- bool **W25qxx_Test** ()

5.48.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2023-08-05

Copyright

5.49 W25Qxx.h

Go to the documentation of this file.

```
00001
00013
00014 #ifndef _W25Qxx_H    /* Guard against multiple inclusion */
00015 #define _W25Qxx_H
00016
00018
00019 #include <stdint.h>
00020 #include <stdbool.h>
00021 #include "spi.h"
00022 #include "gpio.h"
00023
00025
00026 #define W25QXXH_SPI_HANDLE      (&hspi2)
00027 #define W25QXXH_SPI_TIMEOUT_MS  (100)
00028
00029 #define W25QXXH_SPI_CS_PORT     (SPI2_NSS_GPIO_Port)
00030 #define W25XXH_SPI_CS_PIN       (SPI2_NSS_Pin)
00031
00033
00034 #define _W25QXX_DEBUG           (0)
00035
00036 #define DF_MAX_NO_PAGE          65536
00037 #define DF_PAGE_SIZE            256
00038 #define DF_SECTOR_SIZE          16 //16 pages in one sector = 4KB
00039 #define DF_SBLOCK_SIZE          128 //32KB
00040 #define DF_BBLOCK_SIZE          256 //64KB
00041 #define DF_MAX_SECTOR           4096
00042 #define DF_MAX_SBLOCK           512
00043 #define DF_MAX_BBLOCK           256
00044
00046
00047 typedef enum
00048 {
00049     W25Q10=1,
00050     W25Q20,
00051     W25Q40,
00052     W25Q80,
00053     W25Q16,
00054     W25Q32,
00055     W25Q64,
00056     W25Q128,
00057     W25Q256,
00058     W25Q512,
00059 }W25QXX_ID_t;
00060
00062
00063 typedef struct{
00064     uint8_t txbuff[DF_PAGE_SIZE];
00065     uint8_t rxbuff[DF_PAGE_SIZE];
00066     uint8_t txbuff_len;
00067     uint8_t rxbuff_len;
00068     uint8_t rxbuff_readpos;
00069     uint8_t df_mode;
00070
00071 }df_dataparam;
00072
00073 typedef struct
00074 {
00075     W25QXX_ID_t ID;
00076     uint8_t     UniqID[8];
00077     uint16_t    PageSize;
00078     uint32_t    PageCount;
00079     uint32_t    SectorSize;
00080     uint32_t    SectorCount;
00081     uint32_t    BlockSize;
00082     uint32_t    BlockCount;
00083     uint32_t    CapacityInKiloByte;
00084     uint8_t     StatusRegister1;
00085     uint8_t     StatusRegister2;
00086     uint8_t     StatusRegister3;
00087     uint8_t     Lock;
00088 }w25qxx_t;
00089
00090 typedef int32_t eW25qxxStatus;
00091
00093
00094 //###########################################################################
00095 // in Page,Sector and block read/write functions, can put 0 to read maximum bytes
00096 //###########################################################################
00097 eW25qxxStatus       W25qxx_Init(void);
00098
00099 eW25qxxStatus       W25qxx_EraseChip(void);
```

```
00100 eW25qxxStatus       W25qxx_EraseSector(uint32_t SectorAddr);
00101 eW25qxxStatus       W25qxx_EraseBlock(uint32_t BlockAddr);
00102
00103 uint32_t    W25qxx_PageToSector(uint32_t PageAddress);
00104 uint32_t    W25qxx_PageToBlock(uint32_t PageAddress);
00105 uint32_t    W25qxx_SectorToBlock(uint32_t SectorAddress);
00106 uint32_t    W25qxx_SectorToPage(uint32_t SectorAddress);
00107 uint32_t    W25qxx_BlockToPage(uint32_t BlockAddress);
00108
00109 bool        W25qxx_IsEmptyPage(uint32_t Page_Address, uint32_t OffsetInByte, uint32_t
      NumByteToCheck_up_to_PageSize);
00110 bool        W25qxx_IsEmptySector(uint32_t Sector_Address, uint32_t OffsetInByte, uint32_t
      NumByteToCheck_up_to_SectorSize);
00111 bool        W25qxx_IsEmptyBlock(uint32_t Block_Address, uint32_t OffsetInByte, uint32_t
      NumByteToCheck_up_to_BlockSize);
00112
00113 eW25qxxStatus       W25qxx_WriteByte(uint8_t pBuffer, uint32_t WriteAddr_inBytes);
00114 eW25qxxStatus       W25qxx_WritePage(uint8_t *pBuffer, uint32_t Page_Address, uint32_t OffsetInByte,
      uint32_t NumByteToWrite_up_to_PageSize);
00115 eW25qxxStatus       W25qxx_WriteSector(uint8_t *pBuffer, uint32_t Sector_Address, uint32_t
      OffsetInByte, uint32_t NumByteToWrite_up_to_SectorSize);
00116 eW25qxxStatus       W25qxx_WriteBlock(uint8_t* pBuffer, uint32_t Block_Address, uint32_t OffsetInByte,
      uint32_t NumByteToWrite_up_to_BlockSize);
00117
00118 eW25qxxStatus       W25qxx_ReadByte(uint8_t *pBuffer, uint32_t Bytes_Address);
00119 eW25qxxStatus       W25qxx_ReadBytes(uint8_t *pBuffer, uint32_t ReadAddr, uint32_t NumByteToRead);
00120 eW25qxxStatus       W25qxx_ReadPage(uint8_t *pBuffer, uint32_t Page_Address,uint32_t OffsetInByte,
      uint32_t NumByteToRead_up_to_PageSize);
00121 eW25qxxStatus       W25qxx_ReadSector(uint8_t *pBuffer, uint32_t Sector_Address,uint32_t OffsetInByte,
      uint32_t NumByteToRead_up_to_SectorSize);
00122 eW25qxxStatus       W25qxx_ReadBlock(uint8_t* pBuffer, uint32_t Block_Address, uint32_t OffsetInByte,
      uint32_t NumByteToRead_up_to_BlockSize);
00123
00124 bool W25qxx_Test();
00125
00127
00128 #endif /* _W25Qxx_H */
00129
```

5.50 AppSD_API.c File Reference

```
#include <assert.h>
#include <string.h>
#include ¨AppSD_API.h¨
#include ¨AppConfiguration.h¨
```

Functions

- static sSDFS_t * SDFs_GetInstance ()
- static eStorageFSStatus_t SDFs_Init (sSDFS_t *const pMe)
- __attribute__ ((unused))
- static eStorageFSStatus_t SDFs_OpenFileRaw (sSDFS_t *const pMe, eStorageFileNamesEnums_t file↩ Enum, eStorageFSOperationModes_t modeEnum)
- static eStorageFSStatus_t SDFs_CloseFileRaw (sSDFS_t *const pMe, eStorageFileNamesEnums_t file↩ Enum)
- static eStorageFSStatus_t SDFs_ReadFileRaw (sSDFS_t *const pMe, eStorageFileNamesEnums_t file↩ Enum, char *const pOutReadBuf, uint32_t bytesToRead, uint32_t *const pOutBytesRead)
- static eStorageFSStatus_t SDFs_ComputeFileCRC (sSDFS_t *const pMe, eStorageFileNamesEnums_t fileEnum, uint8_t *const pInOutRamBuf, uint32_t RamBufSize, uint32_t *const pOutCRC)
- static eStorageFSStatus_t SDFs_GetFileSizeRaw (const sSDFS_t *const pMe, eStorageFileNamesEnums_t fileEnum, uint32_t *const pOutFileSizeInBytes)
- static eStorageFSStatus_t SDFs_GetFilePresent (sSDFS_t *const pMe, eStorageFileNamesEnums_t file↩ Enum)
- eStorageFSStatus_t SDFs_API_Init ()
- eStorageFSStatus_t SDFs_API_GetGoldenFileStatus ()

- eStorageFSStatus_t SDFs_API_OpenGoldenImageFile ()
- eStorageFSStatus_t SDFs_API_GetGoldenImageFileSize (uint32_t *pOutFileSizeInBytes)
- eStorageFSStatus_t SDFs_API_ReadGoldenImageFile (char *const pOutReadBuf, uint32_t bytesToRead, uint32_t *const pOutBytesRead)
- eStorageFSStatus_t SDFs_API_CloseGoldenImageFile ()
- eStorageFSStatus_t SDFs_API_ComputeGoldenImageFileCRC (uint8_t *const pInOutRamBuf, uint32_t RamBufSize, uint32_t *const pOutCRC)

Variables

- static const uint8_t gcOpModeToFatFSModeConverterTable [eFS_MODE_MAX]
- static const char *const gcFilesNamesTable [eFS_MAX]
- static sSDFS_t gSDCard = {.IsMounted = false}

5.50.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2024-06-07

Copyright

Copyright (c) 2024

5.50.2 Function Documentation

5.50.2.1 SDFs_GetInstance()

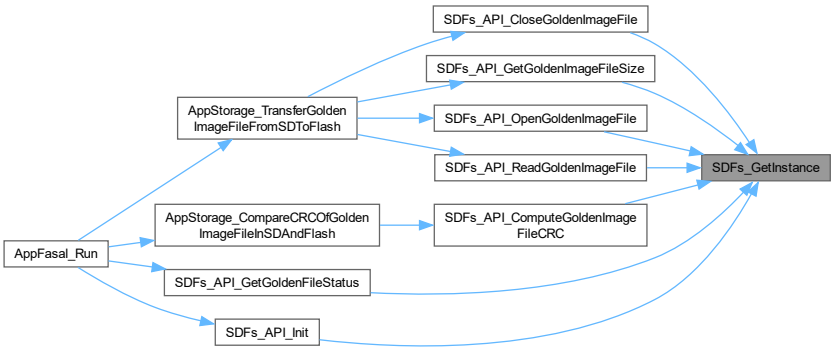
```
static sSDFS_t * SDFs_GetInstance ( )  [static]
```

Get storage media instance.

Returns

sSDFS_t*

Here is the caller graph for this function:



5.50.2.2 SDFs_Init()

```
static eStorageFSStatus_t SDFs_Init (
            sSDFS_t *const pMe )  [static]
```

Initialize FatFS file-system.

Parameters

pMe	FatFS wrapper instance

Returns

eStorageFSStatus_t

Here is the caller graph for this function:

5.50.2.3 __attribute__()

```
__attribute__ (
            (unused)  )
```

Initialize FatFS file-system.

Delete File.

Write to a file in FatFS.

Parameters

pMe	FatFS wrapper instance

Returns

eStorageFSStatus_t

Parameters

pMe	FatFS wrapper instance
fileEnum	File to be written into
IsAppend	Pass true to write to file in append mode, false to write at the beginning
pInWriteBuf	data to be written is passed here
bufSize	size of buffer passed for writing

Returns

eStorageFSStatus_t

Parameters

pMe	FatFS wrapper instance
fileEnum	File to be deleted

Returns

eStorageFSStatus_t

5.50.2.4 SDFs_OpenFileRaw()

```
static eStorageFSStatus_t SDFs_OpenFileRaw (
            sSDFS_t *const pMe,
            eStorageFileNamesEnums_t fileEnum,
            eStorageFSOperationModes_t modeEnum )  [static]
```

Open a file in FatFS.

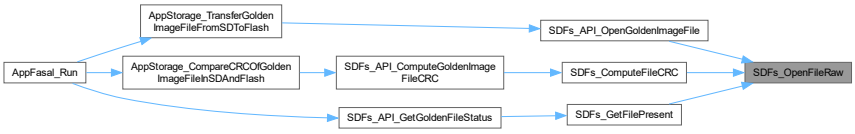
Parameters

pMe	FatFS wrapper instance
fileEnum	File to be opened
modeEnum	Mode to open file In

Returns

eStorageFSStatus_t

Here is the caller graph for this function:



5.50.2.5 SDFs_CloseFileRaw()

```
static eStorageFSStatus_t SDFs_CloseFileRaw (
            sSDFS_t *const pMe,
            eStorageFileNamesEnums_t fileEnum )  [static]
```

Close a file in FatFS.

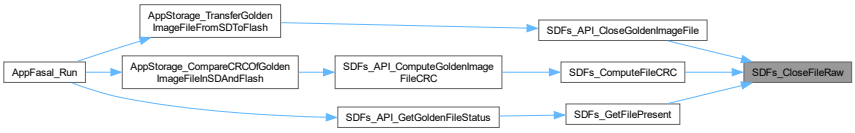
Parameters

pMe	FatFS wrapper instance
fileEnum	file to be closed

Returns

eStorageFSStatus_t

Here is the caller graph for this function:



5.50.2.6 SDFs_ReadFileRaw()

```
static eStorageFSStatus_t SDFs_ReadFileRaw (
            sSDFS_t *const pMe,
            eStorageFileNamesEnums_t fileEnum,
            char *const pOutReadBuf,
            uint32_t bytesToRead,
            uint32_t *const pOutBytesRead )  [static]
```

read from a file in FatFS

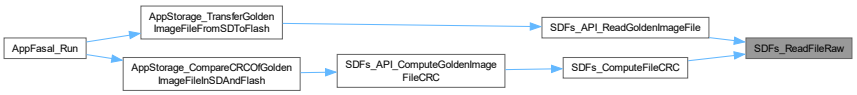
Parameters

pMe	FatFS wrapper instance
fileEnum	File to be read from
pOutReadBuf	read data will be saved here
bytesToRead	number of bytes to read
pOutBytesRead	number of bytes read is saved here

Returns

eStorageFSStatus_t

Here is the caller graph for this function:



5.50.2.7 SDFs_ComputeFileCRC()

```
static eStorageFSStatus_t SDFs_ComputeFileCRC (
            sSDFS_t *const pMe,
            eStorageFileNamesEnums_t fileEnum,
            uint8_t *const pInOutRamBuf,
            uint32_t RamBufSize,
            uint32_t *const pOutCRC )  [static]
```

compute CRC of requested file

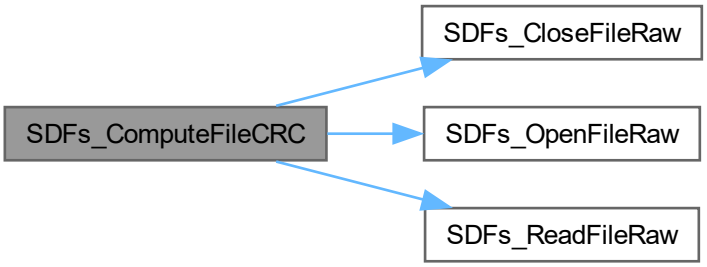
Parameters

pMe	FatFS wrapper instance
fileEnum	File whose CRC needs to be computed
pInOutRamBuf	Ram buffer that will be used as temporary storage while computing CRC
RamBufSize	ram buffer size passed
pOutCRC	computed CRC will be saved here

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.50.2.8 SDFs_GetFileSizeRaw()

```
static eStorageFSStatus_t SDFs_GetFileSizeRaw (
            const sSDFS_t *const pMe,
            eStorageFileNamesEnums_t fileEnum,
            uint32_t *const pOutFileSizeInBytes )  [static]
```

Get size of file in FatFS.

Parameters

pMe	FatFS wrapper instance
fileEnum	file whose size needs to be determined
pOutFileSizeInBytes	size of the file computed is saved here

Returns

eStorageFSStatus_t

Here is the caller graph for this function:

5.50.2.9 SDFs_GetFilePresent()

```
static eStorageFSStatus_t SDFs_GetFilePresent (
            sSDFS_t *const pMe,
            eStorageFileNamesEnums_t fileEnum )  [static]
```

Determine if a file is present in FatFS.

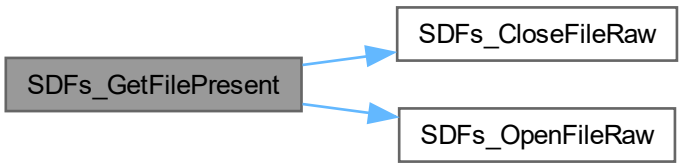
Parameters

pMe	FatFS wrapper instance
fileEnum	file to be checked

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.50.2.10 SDFs_API_Init()

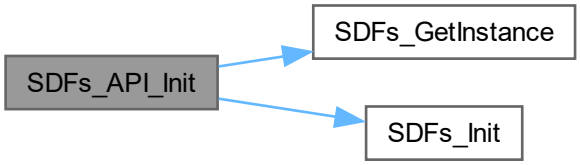
```
eStorageFSStatus_t SDFs_API_Init ( )
```

Initialize FatFS wrapper.

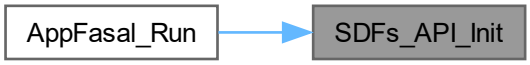
Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:



5.50.2.11 SDFs_API_GetGoldenFileStatus()

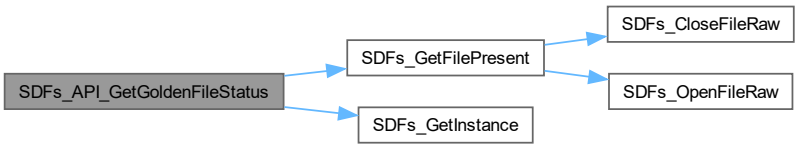
```
eStorageFSStatus_t SDFs_API_GetGoldenFileStatus ( )
```

Check if file exists.

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.50.2.12 SDFs_API_OpenGoldenImageFile()

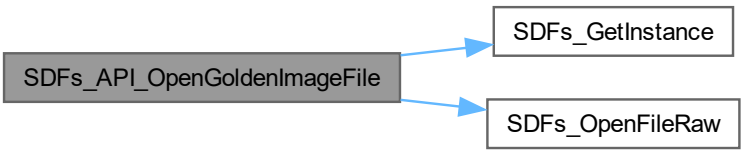
```
eStorageFSStatus_t SDFs_API_OpenGoldenImageFile ( )
```

Open Golden Image file.

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.50.2.13 SDFs_API_GetGoldenImageFileSize()

```
eStorageFSStatus_t SDFs_API_GetGoldenImageFileSize (
            uint32_t * pOutFileSizeInBytes )
```

Get size of golden image.

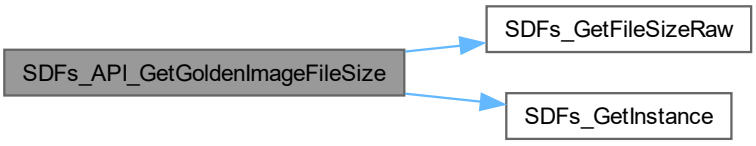
Parameters

pOutFileSizeInBytes	size of the golden image is saved here

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.50.2.14 SDFs_API_ReadGoldenImageFile()

```
eStorageFSStatus_t SDFs_API_ReadGoldenImageFile (
            char *const pOutReadBuf,
            uint32_t bytesToRead,
            uint32_t *const pOutBytesRead )
```

Read golden image file.

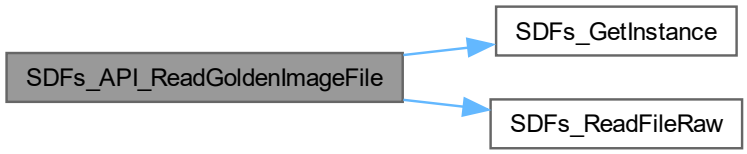
Parameters

pOutReadBuf	contents of read file are saved here
bytesToRead	bytes to read from file
pOutBytesRead	bytes read from file

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.50.2.15 SDFs_API_CloseGoldenImageFile()

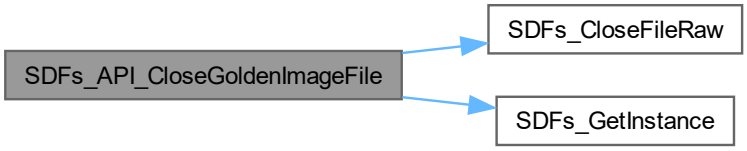
```
eStorageFSStatus_t SDFs_API_CloseGoldenImageFile ( )
```

Close golden Image file.

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.50.2.16 SDFs_API_ComputeGoldenImageFileCRC()

```
eStorageFSStatus_t SDFs_API_ComputeGoldenImageFileCRC (
            uint8_t *const pInOutRamBuf,
            uint32_t RamBufSize,
            uint32_t *const pOutCRC )
```

compute CRC of golden Image file

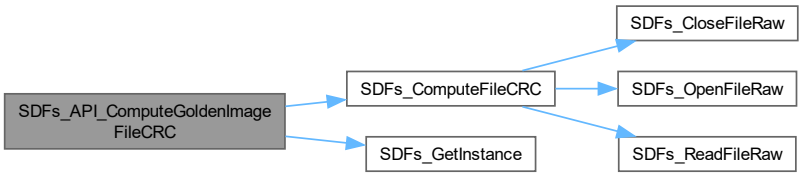
Parameters

pInOutRamBuf	Ram buffer that will be used as temporary storage while computing CRC
RamBufSize	ram buffer size passed
pOutCRC	computed CRC will be saved here

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.50.3 Variable Documentation

5.50.3.1 gcOpModeToFatFSModeConverterTable

```
const uint8_t gcOpModeToFatFSModeConverterTable[eFS_MODE_MAX]  [static]
```

Initial value:

```
=
{
        [eFS_READONLY]          = (FA_READ),
        [eFS_WRITEONLY]         = (FA_WRITE),
        [eFS_READ_WRITE]        = (FA_READ  | FA_WRITE),
        [eFS_READ_CREATE]       = (FA_READ  | FA_OPEN_ALWAYS),
        [eFS_WRITE_CREATE]      = (FA_WRITE | FA_OPEN_ALWAYS),
        [eFS_WRITE_APPEND]      = (FA_WRITE | FA_OPEN_ALWAYS),
        [eFS_READ_WRITE_CREATE] = (FA_READ  | FA_WRITE | FA_OPEN_ALWAYS),
}
```

Utility table to convert user file-system modes to FatFS file system modes.

5.50.3.2 gcFilesNamesTable

```
const char* const gcFilesNamesTable[eFS_MAX]  [static]
```

Initial value:

```
=
{
        [eFS_GOLDEN_IMAGE]  = "fallback.txt",
        [eFS_FILE_2]        = "file2.txt"
}
```

Map Name enums to File name strings.

5.50.3.3 gSDCard

```
sSDFS_t gSDCard = {.IsMounted = false}  [static]
```

Global SD-card instance

5.51 AppSD_API.h File Reference

```
#include <stdbool.h>
#include ¨crc.h¨
#include ¨ff.h¨
#include ¨AppStorageDataStructures.h¨
```

Data Structures

- struct sSDFS_t

Macros

- #define SDFS_CRC_INSTANCE (&hcrc)

Functions

- eStorageFSStatus_t SDFs_API_Init ()
- eStorageFSStatus_t SDFs_API_GetGoldenFileStatus ()
- eStorageFSStatus_t SDFs_API_OpenGoldenImageFile ()
- eStorageFSStatus_t SDFs_API_GetGoldenImageFileSize (uint32_t *pOutFileSizeInBytes)
- eStorageFSStatus_t SDFs_API_ReadGoldenImageFile (char *const pOutReadBuf, uint32_t bytesToRead, uint32_t *const pOutBytesRead)
- eStorageFSStatus_t SDFs_API_CloseGoldenImageFile ()
- eStorageFSStatus_t SDFs_API_ComputeGoldenImageFileCRC (uint8_t *const pInOutRamBuf, uint32_t RamBufSize, uint32_t *const pOutCRC)

5.51.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2024-06-07

Copyright

5.51.2 Macro Definition Documentation

5.51.2.1 SDFS_CRC_INSTANCE

```
#define SDFS_CRC_INSTANCE (&hcrc)
```

CRC instance used by SD-Card module for file integrity check

5.51.3 Function Documentation

5.51.3.1 SDFs_API_Init()

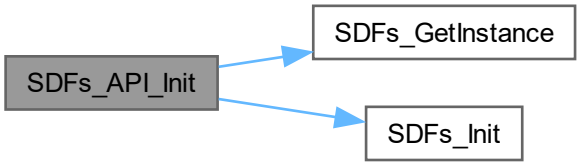
```
eStorageFSStatus_t SDFs_API_Init ( )
```

Initialize FatFS wrapper.

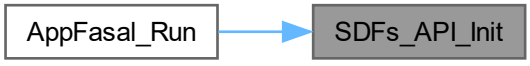
Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:



5.51.3.2 SDFs_API_GetGoldenFileStatus()

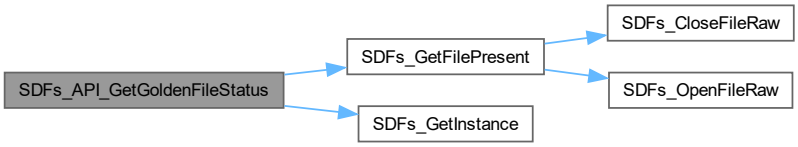
```
eStorageFSStatus_t SDFs_API_GetGoldenFileStatus ( )
```

Check if file exists.

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.51.3.3 SDFs_API_OpenGoldenImageFile()

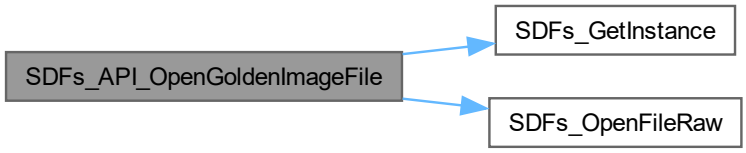
```
eStorageFSStatus_t SDFs_API_OpenGoldenImageFile ( )
```

Open Golden Image file.

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.51.3.4 SDFs_API_GetGoldenImageFileSize()

```
eStorageFSStatus_t SDFs_API_GetGoldenImageFileSize (
            uint32_t * pOutFileSizeInBytes )
```

Get size of golden image.

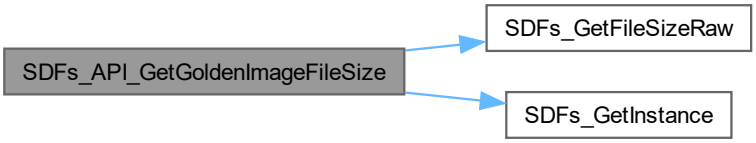
Parameters

pOutFileSizeInBytes	size of the golden image is saved here

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.51.3.5 SDFs_API_ReadGoldenImageFile()

```
eStorageFSStatus_t SDFs_API_ReadGoldenImageFile (
            char *const pOutReadBuf,
            uint32_t bytesToRead,
            uint32_t *const pOutBytesRead )
```

Read golden image file.

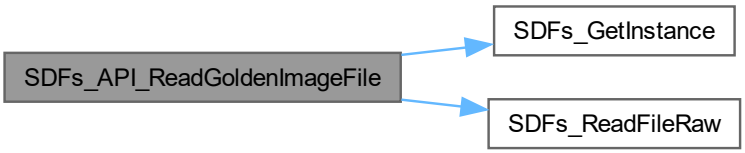
Parameters

pOutReadBuf	contents of read file are saved here
bytesToRead	bytes to read from file
pOutBytesRead	bytes read from file

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.51.3.6 SDFs_API_CloseGoldenImageFile()

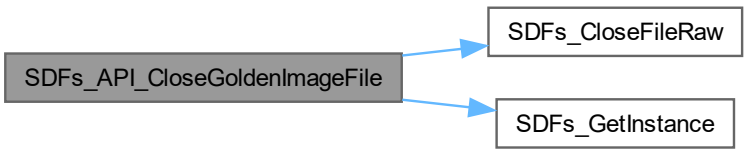
```
eStorageFSStatus_t SDFs_API_CloseGoldenImageFile ( )
```

Close golden Image file.

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.51.3.7 SDFs_API_ComputeGoldenImageFileCRC()

```
eStorageFSStatus_t SDFs_API_ComputeGoldenImageFileCRC (
            uint8_t *const pInOutRamBuf,
            uint32_t RamBufSize,
            uint32_t *const pOutCRC )
```

compute CRC of golden Image file

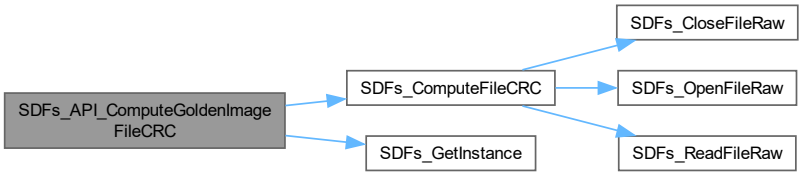
Parameters

pInOutRamBuf	Ram buffer that will be used as temporary storage while computing CRC
RamBufSize	ram buffer size passed
pOutCRC	computed CRC will be saved here

Returns

eStorageFSStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.52 AppSD_API.h

Go to the documentation of this file.

```
00001
00013
00014 #ifndef APPSTORAGE_APPSDFS_APPSD_API_H_
00015 #define APPSTORAGE_APPSDFS_APPSD_API_H_
00016
00017 #include <stdbool.h>
00018 #include "crc.h"
00019 #include "ff.h"
00020 #include "AppStorageDataStructures.h"
00021
00023
00024 #define SDFS_CRC_INSTANCE   (&hcrc)
00027
00032 typedef struct
00033 {
00034     bool IsMounted;
00035     FATFS fs;
00036     FIL fileHandles[eFS_MAX];
00037 }sSDFS_t;
00038
00040
00041 eStorageFSStatus_t SDFs_API_Init();
00042 eStorageFSStatus_t SDFs_API_GetGoldenFileStatus();
00043 eStorageFSStatus_t SDFs_API_OpenGoldenImageFile();
00044 eStorageFSStatus_t SDFs_API_GetGoldenImageFileSize(uint32_t* pOutFileSizeInBytes);
00045 eStorageFSStatus_t SDFs_API_ReadGoldenImageFile(char* const pOutReadBuf, uint32_t bytesToRead,
      uint32_t* const pOutBytesRead);
00046 eStorageFSStatus_t SDFs_API_CloseGoldenImageFile();
00047 eStorageFSStatus_t SDFs_API_ComputeGoldenImageFileCRC(uint8_t* const pInOutRamBuf, uint32_t
      RamBufSize, uint32_t* const pOutCRC);
00048
00050
00051
00052 #endif /* APPSTORAGE_APPSDFS_APPSD_API_H_ */
```

5.53 AppStorage.c File Reference

```
#include <assert.h>
#include <stdbool.h>
#include "SPI.h"
#include "AppStorage.h"
#include "Console.h"
#include "W25Qxx.h"
#include "AppStorageDataStructures.h"
#include "AppSD_API.h"
#include "AppFlash_API.h"
```

Functions

- static __attribute__ ((aligned(4)))
- void AppStorage_SetPower (bool IsEnable)
- eTransferMode_t AppStorage_GetCurrentTransferMode ()
- eStorageFSStatus_t AppStorage_TransferGoldenImageFileFromSDToFlash ()
- eStorageFSStatus_t AppStorage_CompareCRCOfGoldenImageFileInSDAndFlash (bool *const pOutIs↩ CRCMatching)

Variables

- static const eTransferMode_t gcTransferModeToGPIOStatusMappingTable [eTX_MODE_MAX]

5.53.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2024-06-07

Copyright

Copyright (c) 2024

5.53.2 Function Documentation

5.53.2.1 AppStorage_SetPower()

```
void AppStorage_SetPower (
            bool IsEnable )
```

Set Sensor power for powering external flash board.

Note

SPI also needs to be controlled here since power on SPI pins is also powering the external board

Here is the caller graph for this function:

5.53.2.2 AppStorage_GetCurrentTransferMode()

```
eTransferMode_t AppStorage_GetCurrentTransferMode ( )
```

Get current transfer mode setting.

Returns

eTransferMode_t

Here is the caller graph for this function:

5.53.2.3 AppStorage_TransferGoldenImageFileFromSDToFlash()

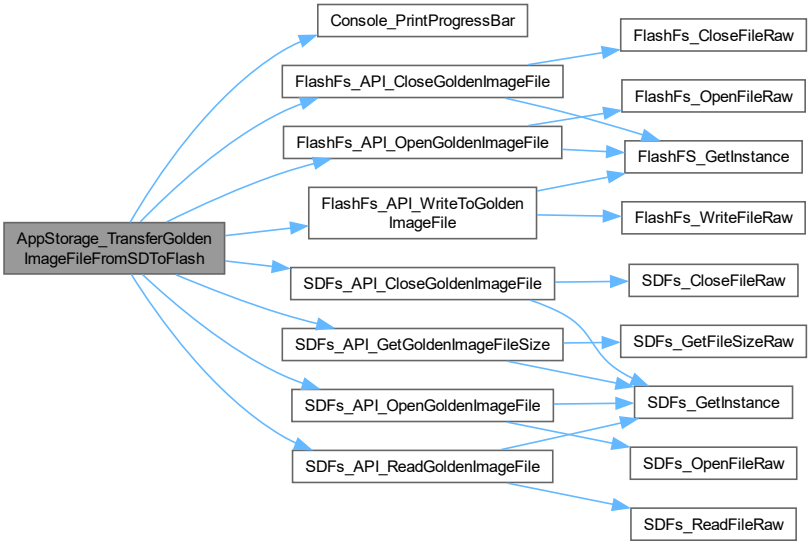
```
eStorageFSStatus_t AppStorage_TransferGoldenImageFileFromSDToFlash ( )
```

Transfer Golden Image from SD card to flash.

Returns

eAppStorageStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.53.2.4 AppStorage_CompareCRCOfGoldenImageFileInSDAndFlash()

```
eStorageFSStatus_t AppStorage_CompareCRCOfGoldenImageFileInSDAndFlash (
            bool *const pOutIsCRCMatching )
```

Compute and compare CRC of golden Image file stored in CRC and Flash.

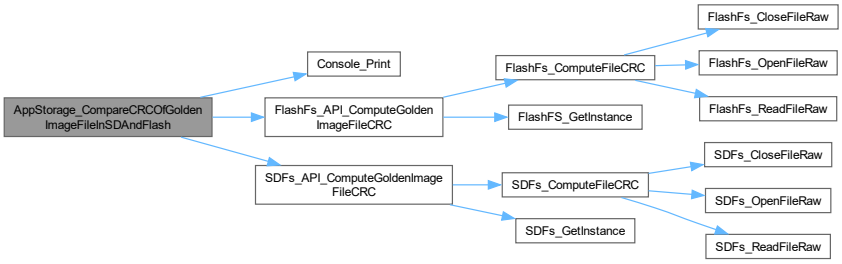
Parameters

pOutIsCRCMatching	Set to true if CRC matches, False otherwise.

Returns

eAppStorageStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.53.3 Variable Documentation

5.53.3.1 gcTransferModeToGPIOStatusMappingTable

```
const eTransferMode_t gcTransferModeToGPIOStatusMappingTable[eTX_MODE_MAX]  [static]
```

Initial value:

```
=
{
        [eTX_MODE_SDCARD_TO_FLASH] = GPIO_PIN_RESET,
        [eTX_MODE_XMODEM_TO_FLASH] = GPIO_PIN_SET
}
```

utility mapping of Transfer mode eTransferMode_t to GPIO state

48k Ram buffer chunks to read the file into, must be aligned to prevent alignment fault

5.54 AppStorage.h File Reference

```
#include <stdbool.h>
#include ¨AppStorageDataStructures.h¨
```

Macros

- #define FLASH_POWER_EN_PORT (SENSOR_POWER_ENABLE_GPIO_Port)
- #define FLASH_POWER_EN_PIN (SENSOR_POWER_ENABLE_Pin)
- #define TRANSFER_MODE_PORT (TRANSFER_MODE_GPIO_Port)
- #define TRANSFE_MODE_PIN (TRANSFER_MODE_Pin)

Enumerations

- enum eTransferMode_t { **eTX_MODE_SDCARD_TO_FLASH** , **eTX_MODE_XMODEM_TO_FLASH** , **eTX↩ _MODE_MAX** }

Functions

- void AppStorage_SetPower (bool IsEnable)
- eStorageFSStatus_t AppStorage_TransferGoldenImageFileFromSDToFlash ()
- eStorageFSStatus_t AppStorage_CompareCRCOfGoldenImageFileInSDAndFlash (bool *const pOutIs↩ CRCMatching)
- eTransferMode_t AppStorage_GetCurrentTransferMode ()

5.54.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2024-06-07

Copyright

5.54.2 Macro Definition Documentation

5.54.2.1 FLASH_POWER_EN_PORT

```
#define FLASH_POWER_EN_PORT (SENSOR_POWER_ENABLE_GPIO_Port)
```

Port of GPIO that controls power to external board

5.54.2.2 FLASH_POWER_EN_PIN

```
#define FLASH_POWER_EN_PIN (SENSOR_POWER_ENABLE_Pin)
```

Pin of GPIO that controls power to external board

5.54.2.3 TRANSFER_MODE_PORT

```
#define TRANSFER_MODE_PORT (TRANSFER_MODE_GPIO_Port)
```

Port of GPIO that determines the transfer mode of operation

5.54.2.4 TRANSFE_MODE_PIN

```
#define TRANSFE_MODE_PIN (TRANSFER_MODE_Pin)
```

Pin of GPIO that determines the transfer mode of operation

5.54.3 Enumeration Type Documentation

5.54.3.1 eTransferMode_t

```
enum eTransferMode_t
```

Enumerations for storage mediums supported.

5.54.4 Function Documentation

5.54.4.1 AppStorage_SetPower()

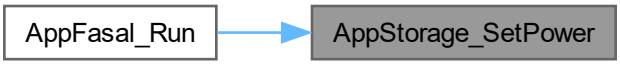
```
void AppStorage_SetPower (
            bool IsEnable )
```

Set Sensor power for powering external flash board.

Note

SPI also needs to be controlled here since power on SPI pins is also powering the external board

Here is the caller graph for this function:



5.54.4.2 AppStorage_TransferGoldenImageFileFromSDToFlash()

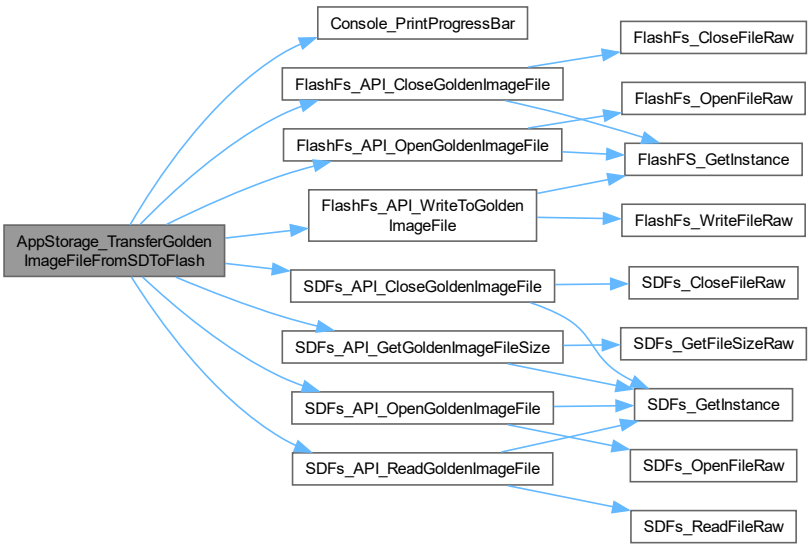
```
eStorageFSStatus_t AppStorage_TransferGoldenImageFileFromSDToFlash ( )
```

Transfer Golden Image from SD card to flash.

Returns

eAppStorageStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:

5.54.4.3 AppStorage_CompareCRCOfGoldenImageFileInSDAndFlash()

```
eStorageFSStatus_t AppStorage_CompareCRCOfGoldenImageFileInSDAndFlash (
            bool *const pOutIsCRCMatching )
```

Compute and compare CRC of golden Image file stored in CRC and Flash.

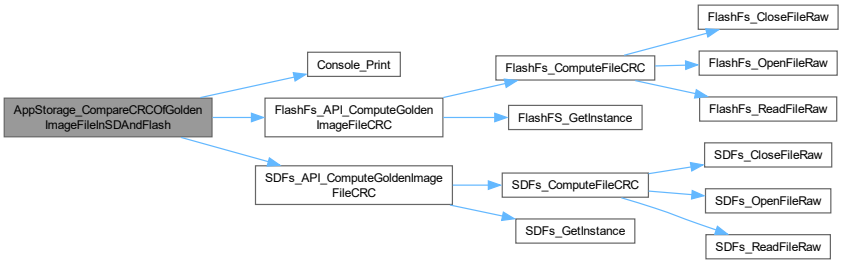
Parameters

pOutIsCRCMatching	Set to true if CRC matches, False otherwise.

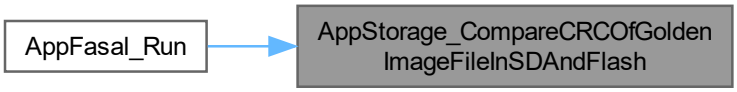
Returns

eAppStorageStatus_t

Here is the call graph for this function:



Here is the caller graph for this function:



5.54.4.4 AppStorage_GetCurrentTransferMode()

```
eTransferMode_t AppStorage_GetCurrentTransferMode ( )
```

Get current transfer mode setting.

Returns

eTransferMode_t

Here is the caller graph for this function:

5.55 AppStorage.h

Go to the documentation of this file.

```
00001
00012
00013 #ifndef APPSTORAGE_APPSTORAGE_H_
00014 #define APPSTORAGE_APPSTORAGE_H_
00015
00017
00018 #include <stdbool.h>
00019 #include "AppStorageDataStructures.h"
00020
00022
00023 #define FLASH_POWER_EN_PORT (SENSOR_POWER_ENABLE_GPIO_Port)
00024 #define FLASH_POWER_EN_PIN  (SENSOR_POWER_ENABLE_Pin)
00026 #define TRANSFER_MODE_PORT  (TRANSFER_MODE_GPIO_Port)
00027 #define TRANSFE_MODE_PIN    (TRANSFER_MODE_Pin)
00030
00035 typedef enum
00036 {
00037     eTX_MODE_SDCARD_TO_FLASH,
00038     eTX_MODE_XMODEM_TO_FLASH,
00039     eTX_MODE_MAX
00040 }eTransferMode_t;
00041
00043
00044 void AppStorage_SetPower(bool IsEnable);
00045 eStorageFSStatus_t AppStorage_TransferGoldenImageFileFromSDToFlash();
00046 eStorageFSStatus_t AppStorage_CompareCRCOfGoldenImageFileInSDAndFlash(bool* const pOutIsCRCMatching);
00047 eTransferMode_t AppStorage_GetCurrentTransferMode();
00048
00050
00051 #endif /* APPSTORAGE_APPSTORAGE_H_ */
```

5.56 AppStorageDataStructures.h File Reference

Enumerations

- enum eStorageFSStatus_t { **eFS_SUCCESS** , **eFS_ERROR** , **eFS_NO_FILE** , **eFS_MAX_STATUS** }
- enum eStorageFSOperationModes_t { **eFS_READONLY** , **eFS_WRITEONLY** , **eFS_READ_WRITE** , **eFS_READ_CREATE** , **eFS_WRITE_CREATE** , **eFS_WRITE_APPEND** , **eFS_READ_WRITE_CREATE** , **eFS_MODE_MAX** }
- enum eStorageFileNamesEnums_t { **eFS_GOLDEN_IMAGE** , **eFS_FILE_2** , **eFS_MAX** }

5.56.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2024-06-07

Copyright

5.56.2 Enumeration Type Documentation

5.56.2.1 eStorageFSStatus_t

enum eStorageFSStatus_t

Storage medium statuses defined here.

5.56.2.2 eStorageFSOperationModes_t

enum eStorageFSOperationModes_t

User enumeration RW modes for file system.

5.56.2.3 eStorageFileNamesEnums_t

enum eStorageFileNamesEnums_t

Enums for all the files to be saved in Storage medium.

5.57 AppStorageDataStructures.h

Go to the documentation of this file.

```
00001
00013
00014 #ifndef APPSTORAGE_APPSTORAGEDATASTRUCTURES_H_
00015 #define APPSTORAGE_APPSTORAGEDATASTRUCTURES_H_
00016
00018
00023 typedef enum
00024 {
00025     eFS_SUCCESS,
00026     eFS_ERROR,
00027     eFS_NO_FILE,
00028     eFS_MAX_STATUS,
00029 }eStorageFSStatus_t;
00030
00035 typedef enum
00036 {
00037     eFS_READONLY,
00038     eFS_WRITEONLY,
00039     eFS_READ_WRITE,
00040     eFS_READ_CREATE,
00041     eFS_WRITE_CREATE,
00042     eFS_WRITE_APPEND,
00043     eFS_READ_WRITE_CREATE,
00044     eFS_MODE_MAX
00045 }eStorageFSOperationModes_t;
00046
00051 typedef enum
00052 {
00053     eFS_GOLDEN_IMAGE,
00054     eFS_FILE_2,
00055     eFS_MAX
00056 }eStorageFileNamesEnums_t;
00057
00059
00060 #endif /* APPSTORAGE_APPSTORAGEDATASTRUCTURES_H_ */
```

5.58 xmodem.c File Reference

```
#include <string.h>
#include ¨xmodem.h¨
#include ¨Console.h¨
#include ¨AppFlash_API.h¨
```

Functions

- static uint16_t xmodem_computeCRC (uint8_t *data, uint16_t length)
- static xmodem_status xmodem_handlePacket (uint8_t header)
- static xmodem_status xmodem_errorHandler (uint8_t *error_number, uint8_t max_error_number)
- xmodem_status xmodem_API_receive (void)

Variables

- static uint8_t gxModemPacketNumber = 1u
- static uint8_t gxModemIsFirstPacket = false

5.58.1 Detailed Description

Author

Vishal keshava Murthy

Version

0.1

Date

2024-02-22

Copyright

Copyright (c) 2024

Note

: Original module source https://github.com/ferenc-nemeth

5.58.2 Function Documentation

5.58.2.1 xmodem_computeCRC()

```
static uint16_t xmodem_computeCRC (
            uint8_t * data,
            uint16_t length )  [static]
```

Calculates the CRC-16 for the input package.

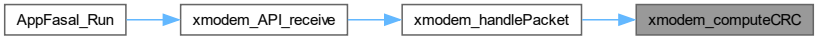
Parameters

∗data	Array of the data which we want to calculate.
length	Size of the data, either 128 or 1024 bytes.

Returns

status: The calculated CRC.

Here is the caller graph for this function:



5.58.2.2 xmodem_handlePacket()

```
static xmodem_status xmodem_handlePacket (
            uint8_t header )  [static]
```

This function handles the data packet we get from the xmodem protocol.

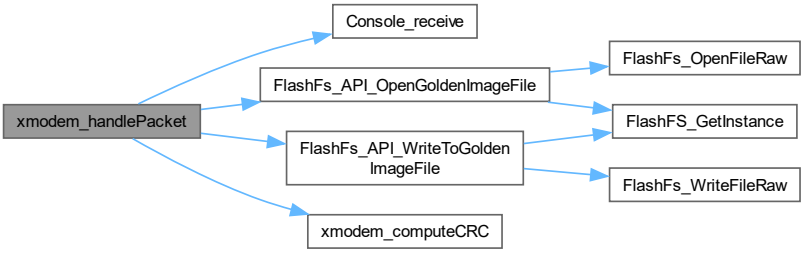
Parameters

header	SOH or STX.

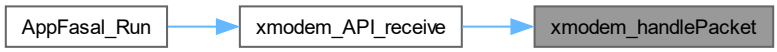
Returns

status: Report about the packet.

Here is the call graph for this function:



Here is the caller graph for this function:



5.58.2.3 xmodem_errorHandler()

```
static xmodem_status xmodem_errorHandler (
            uint8_t * error_number,
            uint8_t max_error_number )  [static]
```

Handles the xmodem error. Raises the error counter, then if the number of the errors reached critical, do a graceful abort, otherwise send a NAK.

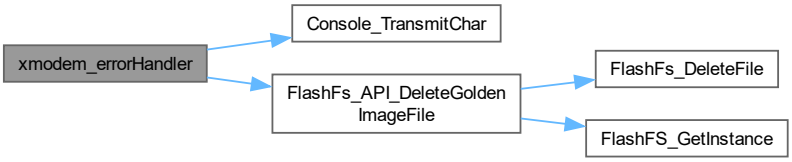
Parameters

∗error_number	Number of current errors (passed as a pointer).
max_error_number	Maximal allowed number of errors.

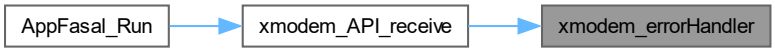
Returns

status: X_ERROR in case of too many errors, X_OK otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



5.58.2.4 xmodem_API_receive()

```
xmodem_status xmodem_API_receive (
            void  )
```

This function is the base of the Xmodem protocol. When we receive a header from UART, it decides what action it shall take.

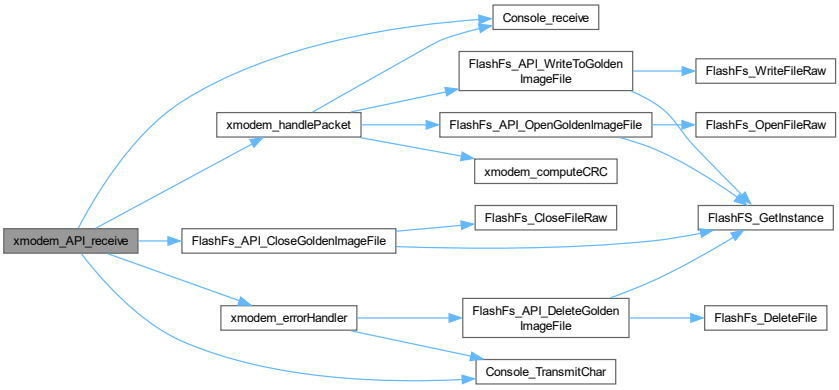
Parameters

void	

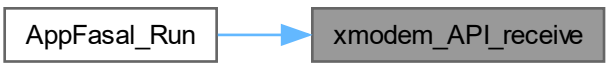
Returns

xmodem_status

Here is the call graph for this function:



Here is the caller graph for this function:



5.58.3 Variable Documentation

5.58.3.1 gxModemPacketNumber

```
uint8_t gxModemPacketNumber = 1u  [static]
```

Packet number counter.

5.58.3.2 gxModemIsFirstPacket

```
uint8_t gxModemIsFirstPacket = false  [static]
```

First packet or not.

5.59 xmodem.h File Reference

```
#include ¨stdbool.h¨
```

Macros

- #define **X_MAX_ERRORS** ((uint8_t)3u)
- #define **X_PACKET_NUMBER_SIZE** ((uint16_t)2u)
- #define **X_PACKET_128_SIZE** ((uint16_t)128u)
- #define **X_PACKET_1024_SIZE** ((uint16_t)1024u)
- #define **X_PACKET_CRC_SIZE** ((uint16_t)2u)
- #define **X_PACKET_NUMBER_INDEX** ((uint16_t)0u)
- #define **X_PACKET_NUMBER_COMPLEMENT_INDEX** ((uint16_t)1u)
- #define **X_PACKET_CRC_HIGH_INDEX** ((uint16_t)0u)
- #define **X_PACKET_CRC_LOW_INDEX** ((uint16_t)1u)
- #define X_SOH ((uint8_t)0x01u)
- #define X_STX ((uint8_t)0x02u)
- #define X_EOT ((uint8_t)0x04u)
- #define X_ACK ((uint8_t)0x06u)
- #define X_NAK ((uint8_t)0x15u)
- #define X_CAN ((uint8_t)0x18u)
- #define X_C ((uint8_t)0x43u)

Enumerations

- enum xmodem_status {
  X_OK = 0x00u , X_ERROR_CRC = 0x01u , X_ERROR_NUMBER = 0x02u , X_ERROR_UART = 0x04u ,
  X_ERROR_FLASH = 0x08u , X_COMPLETE = 0x10u , X_ERROR = 0xFFu }

Functions

- xmodem_status xmodem_API_receive (void)

5.59.1 Detailed Description

Author

Vishal Keshava Murthy

Version

0.1

Date

2024-02-22

Copyright

Note

Original module source https://github.com/ferenc-nemeth

5.59.2 Macro Definition Documentation

5.59.2.1 X_SOH

```
#define X_SOH ((uint8_t)0x01u)
```

Start Of Header (128 bytes).

5.59.2.2 X_STX

```
#define X_STX ((uint8_t)0x02u)
```

Start Of Header (1024 bytes).

5.59.2.3 X_EOT

```
#define X_EOT ((uint8_t)0x04u)
```

End Of Transmission.

5.59.2.4 X_ACK

```
#define X_ACK ((uint8_t)0x06u)
```

Acknowledge.

5.59.2.5 X_NAK

```
#define X_NAK ((uint8_t)0x15u)
```

Not Acknowledge.

5.59.2.6 X_CAN

```
#define X_CAN ((uint8_t)0x18u)
```

Cancel.

5.59.2.7 X_C

```
#define X_C ((uint8_t)0x43u)
```

ASCII ¨C¨ to notify the host we want to use CRC16.

5.59.3 Enumeration Type Documentation

5.59.3.1 xmodem_status

```
enum xmodem_status
```

xModem status enums

Enumerator

X_OK	The action was successful.
X_ERROR_CRC	CRC calculation error.
X_ERROR_NUMBER	Packet number mismatch error.
X_ERROR_UART	UART communication error.
X_ERROR_FLASH	Flash related error.
X_COMPLETE	Generic error.
X_ERROR	Generic error.

5.59.4 Function Documentation

5.59.4.1 xmodem_API_receive()

```
xmodem_status xmodem_API_receive (
            void  )
```

This function is the base of the Xmodem protocol. When we receive a header from UART, it decides what action it shall take.

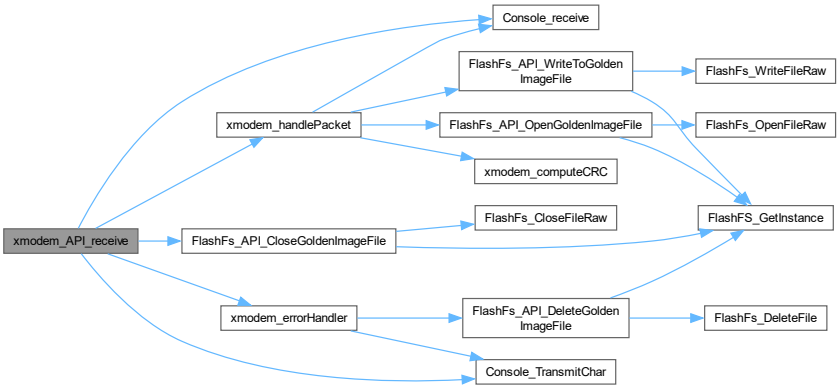
Parameters

void	

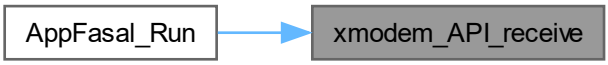
Returns

xmodem_status

Here is the call graph for this function:



Here is the caller graph for this function:



5.60 xmodem.h

Go to the documentation of this file.

```
00001
00012 #ifndef XMODEM_H_
00013 #define XMODEM_H_
00014
00016
00017 #include "stdbool.h"
00018
00020
00021 /* Xmodem (128 bytes) packet format
00022  * Byte  0:       Header
00023  * Byte  1:       Packet number
00024  * Byte  2:       Packet number complement
00025  * Bytes 3-130:   Data
00026  * Bytes 131-132: CRC
00027  */
00028
```

```
00029 /* Xmodem (1024 bytes) packet format
00030  * Byte  0:         Header
00031  * Byte  1:         Packet number
00032  * Byte  2:         Packet number complement
00033  * Bytes 3-1026:    Data
00034  * Bytes 1027-1028: CRC
00035  */
00036
00037 /* Maximum allowed errors (user defined). */
00038 #define X_MAX_ERRORS ((uint8_t)3u)
00039
00040 /* Sizes of the packets. */
00041 #define X_PACKET_NUMBER_SIZE  ((uint16_t)2u)
00042 #define X_PACKET_128_SIZE     ((uint16_t)128u)
00043 #define X_PACKET_1024_SIZE    ((uint16_t)1024u)
00044 #define X_PACKET_CRC_SIZE     ((uint16_t)2u)
00045
00046 /* Indexes inside packets. */
00047 #define X_PACKET_NUMBER_INDEX             ((uint16_t)0u)
00048 #define X_PACKET_NUMBER_COMPLEMENT_INDEX  ((uint16_t)1u)
00049 #define X_PACKET_CRC_HIGH_INDEX           ((uint16_t)0u)
00050 #define X_PACKET_CRC_LOW_INDEX            ((uint16_t)1u)
00051
00052
00053 /* Bytes defined by the protocol. */
00054 #define X_SOH ((uint8_t)0x01u)
00055 #define X_STX ((uint8_t)0x02u)
00056 #define X_EOT ((uint8_t)0x04u)
00057 #define X_ACK ((uint8_t)0x06u)
00058 #define X_NAK ((uint8_t)0x15u)
00059 #define X_CAN ((uint8_t)0x18u)
00060 #define X_C   ((uint8_t)0x43u)
00063
00068 typedef enum {
00069   X_OK            = 0x00u,
00070   X_ERROR_CRC     = 0x01u,
00071   X_ERROR_NUMBER  = 0x02u,
00072   X_ERROR_UART    = 0x04u,
00073   X_ERROR_FLASH   = 0x08u,
00074   X_COMPLETE      = 0x10u,
00075   X_ERROR         = 0xFFu
00076 } xmodem_status;
00077
00079
00080 xmodem_status xmodem_API_receive(void);
00081
00083
00084 #endif /* XMODEM_H_ */
```

Index