eMMC Management Driver User Guide

Version 1.10

For use with eMMC Management Driver versions 1.05 and above

Table of Contents

1. System Overview

This chapter contains the fundamental information for this module.

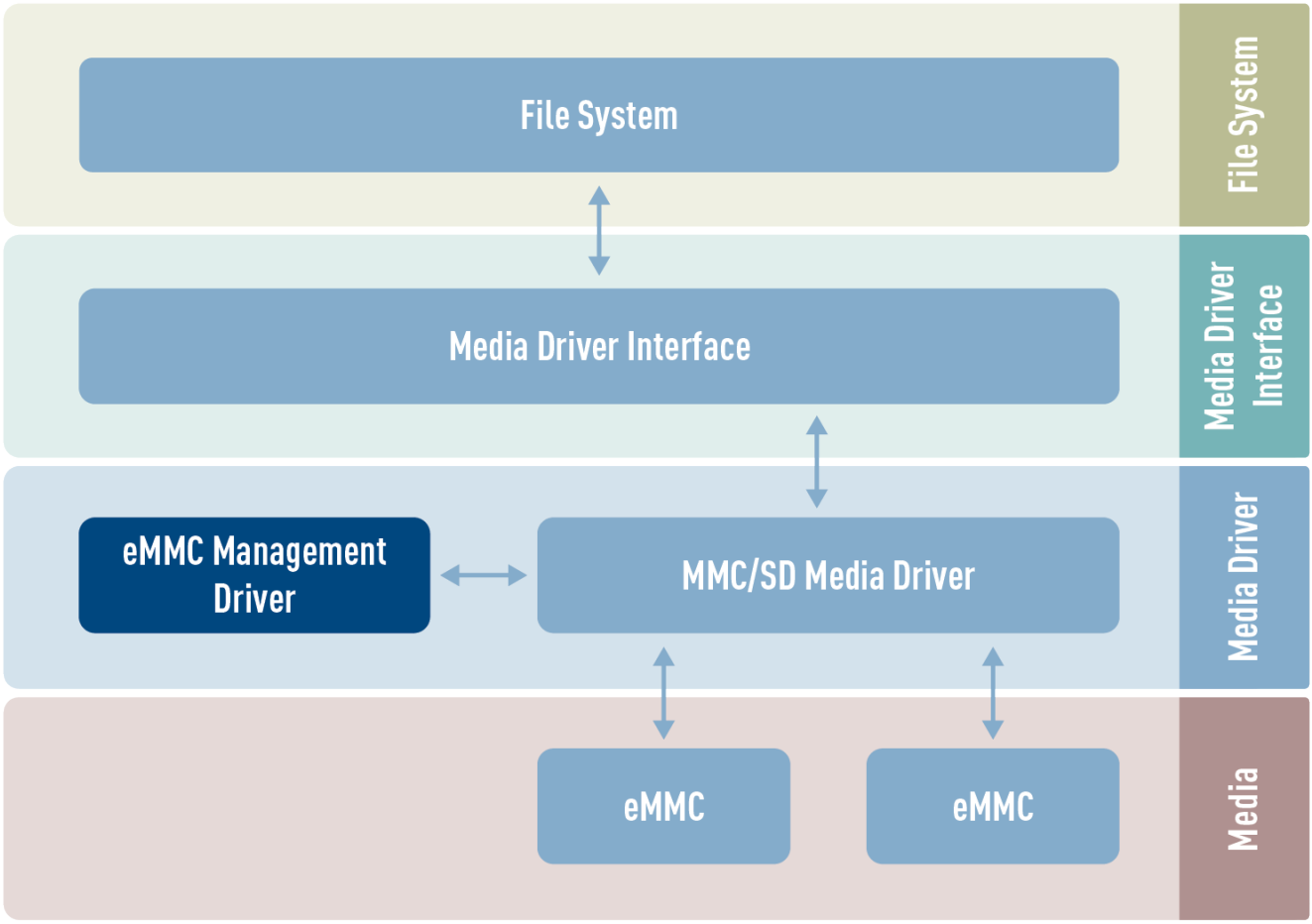
The component sections are as follows:

- Introduction – describes the main elements of the module.
- Feature Check – summarizes the main features of the module as bullet points.
- Packages and Documents – the *Packages* section lists the packages that you need in order to use this module. The *Documents* section lists the relevant user guides.
- Change History – lists the earlier versions of this manual, giving the software version that each manual describes.

1.1. Introduction

This guide is for those who want to use the eMMC (embedded MMC) Management Driver. This driver is an extension to HCC's MultiMedia Card (MMC) and Secure Digital (SD) media drivers and is independent of any particular microcontroller and its MMC/SD controller. This guide covers all aspects of the eMMC Management Driver's configuration and use.

The diagram below shows a typical system architecture including a file system, media driver and media. As an example, this shows both MMC and SD media.



The HCC MMC/SD media drivers conform to the *HCC Media Driver Interface specification*. They provide an interface for a file system to read from/write to MMC or SD storage devices. The media drivers all support eMMC, but the eMMC Management Driver provides these extra capabilities:

- You can obtain card health information including indications of device lifetime based on the average reserved blocks, or on the averaged wear-out of different types of memory.
- You can obtain vendors' proprietary card health reports.
- You can set "partition" configuration for an eMMC, and also obtain this type of information for a card.

Note: Do not confuse partitioning here with partitioning at the file system level. Partitioning at this level effectively creates separate drives that can be uniquely identified by their MMC/SD controller unit ID and the eMMC partition number.

1.2. Feature Check

The main features of the eMMC Management Driver are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Conforms to the *HCC Media Driver Interface Specification*.
- Designed for integration with both RTOS and non-RTOS based systems.
- Provides and lets you set the partition configuration of an eMMC.
- Provides card health information including indications of device lifetime based on the average reserved blocks, or on the averaged wear-out of different types of memory.
- Provides vendors' proprietary card health reports.
- Supports Background Operations (BKOPs).

1.3. Packages and Documents

Packages

The table below lists the packages that you need in order to use this module:

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
media_drv_base	The base media driver package that includes the framework all media drivers use.
media_drv_mmcsd_emmc_management	The eMMC Management Driver package described in this document.

Documents

For an overview of HCC file systems and data storage, see Product Information on the main HCC website.

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC Media Driver Interface Guide

This document describes the HCC Media Driver Interface Specification.

HCC eMMC Management Driver User Guide

This is this document.

1.4. Change History

To download this manual, see File System Media Driver PDFs.

For the history of changes made to the package code itself, see History: media_drv_mmcsd_emmc_management.

The current version of this manual is 1.10.

Manual version	Date	Software version	Reason for change
1.10	2020-04-01	1.05	Added Background Operations (BKOPS) function calls.
1.00	2019-10-04	1.04	First online version.

2. Enabling or Disabling eMMC Management

To enable/disable the eMMC Management Driver, set the option MMCSD_ALLOW_EMMC_MANAGEMENT in the MMCSD media driver module's **config/config_mdriver_mmcsd.h** file.

Set this to 1 to enable the eMMC Management Driver, 1 to disable it.

3. Source File List

This section describes all the source code files included in the system. These files follow the HCC Embedded standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

Note: Do not modify any of these files.

API Header File

The file **src/api/api_mdriver_emmc_management.h** is the only file that should be included by an application using this module. For details of the single API function, see Application Programming Interface.

System Files

These files in the directory **src/media-drv/mmcsd** hold the source code for the media driver. **These files should only be modified by HCC**.

File	Description
emmc_management.c	Source file.
emmc_management.h	Header file.

Version File

The file **src/version/ver_mdriver_emmc_management.h** contains the version number of this module. This version number is checked by all modules that use this module to ensure system consistency over upgrades.

4. Application Programming Interface

This section describes the functions, the main structures used, and the error codes.

4.1. Functions

The functions are the following:

Function	Description
emmc_set_partition_config()	Sets the partition configuration of an eMMC card.
emmc_get_partition_config()	Gets the partition configuration from an eMMC card.
emmc_get_health_status()	Gets the health status of an eMMC card.
emmc_bkops_start()	Writes to the BKOPS_START register to start background operations.
emmc_bkops_set_mode()	Sends a SWITCH command to the card to clear/write bits of the BKOPS_EN register.
emmc_bkops_get_mode()	Reads EXT_CSD and fills the *t_emmc_bkops_mode* structure.
emmc_bkops_get_status()	Reads EXT_CSD and returns the BKOPS_STATUS register.

emmc_set_partition_config

Use this function to set the partition configuration of an eMMC card.

Call this function before initializing any partition on a card. The card is initialized first then released before the function returns, allowing you to call **mmcsd_initfunc()** for any partition or for the User Data Area. If you call this for a card that **mmcsd_initfunc()** has already been called for, it returns MMCSD_ERR_NOTAVAILABLE.

The desired partition settings only contain length information for the four partitions, not for the User Data Area. The User Data Area is allocated the space remaining after the partitions are specified.

Format

```
int emmc_set_partition_config (
    uint32_t                    unit_id,
    t_emmc_partition_setup *    p_part_setup )
```

Arguments

Argument	Description	Type
unit ID	The unit ID of the card to set partition configuration on.	uint32_t
p_part_setup	The partition setup to burn into the card.	t_emmc_partition_setup *

Return values

Return value	Description
MMCSD_NO_ERROR	Successful execution.
Else	See Error Codes.

emmc_get_partition_config

Use this function to get the partition configuration from an eMMC card.

You can call this function whether or not there is at least one initialized partition on the card (in other words, whether or not **mmcsd_initfunc()** has already been called for at least one partition, including the User Data Area, on the card). If **mmcsd_initfunc()** has not been called before this function, the card is initialized first then released before the function returns. This lets you call **mmcsd_initfunc()** for any partition or for the User Data Area.

Format

```
int emmc_get_partition_config (
    uint32_t                        unit_id,
    t_emmc_partition_settings *     p_part_settings )
```

Arguments

Argument	Description	Type
unit ID	The unit ID of the card to get partition configuration from.	uint32_t
p_part_settings	Where to put the partition configuration read from the card.	t_emmc_partition_settings *

Return values

Return value	Description
MMCSD_NO_ERROR	Successful execution.
Else	See Error Codes.

emmc_get_health_status

Use this function to get the health status of an eMMC card.

Note: Call this function before initializing any partition on a card.

Format

```
int emmc_get_health_status (
    uint32_t    unit_id,
    uint32_t    type,
    uint32_t    length,
    uint8_t *   p_dst )
```

Arguments

Argument	Description	Type
unit ID	The unit ID of the card to get health information from.	uint32_t
type	The category of health-related information to obtain.	uint32_t
length	The length of the output buffer.	uint32_t
p_dst	Where to put the information read from the card.	uint8_t *

Return values

Return value	Description
MMCSD_NO_ERROR	Successful execution.
Else	See Error Codes.

emmc_bkops_start

Use this function to write to the BKOPS_START register to start background operations.

Note: This call waits until background operations are finished.

Format

```
int emmc_bkops_start ( uint32_t unit_id )
```

Arguments

Argument	Description	Type
unit ID	The unit ID of the card.	uint32_t

Return values

Return value	Description
MMCSD_NO_ERROR	Successful execution.
Else	See Error Codes.

emmc_bkops_set_mode

Use this function to send a SWITCH command to the card with specific arguments to clear/write bits of the BKOPS_EN register.

Format

```
int emmc_set_mode (
    uint32_t                unit_id,
    t_emmc_bkops_mode *     p_bkops_mode )
```

Arguments

Argument	Description	Type
unit ID	The unit ID of the card to set partition configuration on.	uint32_t
p_bkops_mode	A pointer to the partition setup to burn into the card.	t_emmc_bkops_mode *

Return values

Return value	Description
MMCSD_NO_ERROR	Successful execution.
Else	See Error Codes.

emmc_bkops_get_mode

Use this function to read EXT_CSD and fill the *t_emmc_bkops_mode* structure.

Format

```
int emmc_get_mode (
    uint32_t               unit_id,
    t_emmc_bkops_mode *    p_bkops_mode )
```

Arguments

Argument	Description	Type
unit ID	The unit ID of the card.	uint32_t
p_bkops_mode	A pointer to the structure to fill.	t_emmc_bkops_mode *

Return values

Return value	Description
MMCSD_NO_ERROR	Successful execution.
Else	See Error Codes.

emmc_bkops_get_status

Use this function to read EXT_CSD and return the BKOPS_STATUS register.

Format

```
int emmc_bkops_get_status (
    uint32_t    unit_id,
    uint8_t *   p_status )
```

Arguments

Argument	Description	Type
unit ID	The unit ID of the card.	uint32_t
p_status	A pointer to the status register to fill.	uint8_t *

Return values

Return value	Description
MMCSD_NO_ERROR	Successful execution.
Else	See Error Codes.

4.2. Error Codes

If **mmcsd_initfunc()** executes successfully it returns with MMCSD_NO_ERROR, a value of 0. The following table shows the meaning of the error codes.

Return Value	Value	Description
MMCSD_ERR_NOTPLUGGED	-1	For high level.
MMCSD_NO_ERROR	0	Successful execution.
MMCSD_ERR_NOTINITIALIZED	101	Driver not initialized.
MMCSD_ERR_INIT	102	Initialization error.
MMCSD_ERR_CMD	103	Command error.
MMCSD_ERR_STARTBIT	104	Start bit incorrect.
MMCSD_ERR_BUSY	105	Driver already active.
MMCSD_ERR_CRC	106	CRC error.
MMCSD_ERR_WRITE	107	Write error.
MMCSD_ERR_WRITEPROTECT	108	Media is write-protected.
MMCSD_ERR_DATAOK	109	Data valid.
MMCSD_ERR_RX	110	Receive error.
MMCSD_ERR_NOTAVAILABLE	111	Not available.

4.3. Types and Definitions

This section describes the main elements that are defined in the API Header file.

t_emmc_partition_setup

This structure is used for setting card partitions when calling **emmc_set_partition_config()**.

The length of the user data area cannot be set here as it always has the remaining size after partitions are allocated.

Element	Type	Description
cpsu_part_dsc [EMMC_PARTITION_COUNT]	t_emmc_partition_dsc	The partition descriptor for each partition.
b_cpsu_user_enh	uint8_t	If this is set the User Data Area is enhanced, otherwise it is normal.
b_cpsu_user_rel_wr	uint8_t	If this is set the User Data Area uses reliable write, otherwise it does not.

t_emmc_partition_dsc

This is the partition descriptor, used in the *t_emmc_partition_settings* and *t_emmc_partition_setup* structures:

Element	Type	Description
pdsc_size	uint32_t	The size of the partition in [HC erase group size].
b_pdsc_enh	uint8_t	The partition is enhanced/normal.
b_pdsc_rel_wr	uint8_t	The partition uses reliable write.

t_emmc_partition_settings

The *t_emmc_partition_settings* structure configures a partition:

Element	Type	Description
cpsi_max_enh_size	uint32_t	The maximum size of the enhanced area [HC erase group size].
cpsi_part_dsc [EMMC_PARTITION_COUNT]	t_emmc_partition_dsc	An array with a partition descriptor for each partition.
cpsi_user_dsc	t_emmc_user_data_dsc	The User Data Area descriptor.
cpsi_mul_factor	uint8_t	A multiplication factor for calculating sector number: `sector num = ( xdsc_size * mul_factor )`
b_cpsi_part_supp	uint8_t	This is set if partitions are supported.
b_cpsi_enh_supp	uint8_t	This is set if enhanced features, typically pseudo Single Level Cell (SLC) operation, are supported on partitions.
b_cpsi_part_complt	uint8_t	This is set when partitioning setting is complete.
b_cpsi_rel_wr_supp	uint8_t	This is set if Reliable Write is supported on the partitions and on the User Data Area

t_emmc_user_data_dsc

This is the User Data Area descriptor, used in *t_emmc_partition_settings*:

Element	Type	Description
udsc_size	uint32_t	The size of the User Data Area in [HC erase group size].
b_udsc_enh	uint8_t	The User Data Area is enhanced/normal.
b_udsc_rel_wr	uint8_t	The User Data Area uses reliable write.

Partition Access Definitions

These are defined in the file **src/api/api_mdriver_emmc_management.h**.

EMMC_PARTITION_COUNT is the maximum number of partitions on an eMMC card. This is defined as 4 by the JEDEC standard.

Use the following partition indexing macros with MMCSD_BUILD_PARAMETER:

Macro	Description
EMMC_PARTITION_IDX_USER	Selects the User Data Area.
EMMC_PARTITION_IDX_PARTITION_1	Selects Partition 1.
EMMC_PARTITION_IDX_PARTITION_2	Selects Partition 2.
EMMC_PARTITION_IDX_PARTITION_3	Selects Partition 3.
EMMC_PARTITION_IDX_PARTITION_4	Selects Partition 4.

MMCSD_BUILD_PARAMETER(unit, partition) can be used to create parameter to be used when media driver initialisation function is called.

For example, initialize the User Data Area and Partition2 on eMMC Card 0 using code like this:

```
uint32_t   driver_param;
F_DRIVER * p_drv_usr;
F_DRIVER * p_drv_part2;
    /* Initialise User Data Area */
    driver_param = MMCSD_BUILD_PARAMETER( 0u, EMMC_PARTITION_IDX_USER );
    p_drv_usr = mmcsd_initfunc( driver_param );
    if ( p_drv_usr != NULL )
    {
         /* Initialise Partition2 */
         driver_param = MMCSD_BUILD_PARAMETER( 0u, EMMC_PARTITION_IDX_PARTITION_2 );
         p_drv_part2 = mmcsd_initfunc( driver_param );
         if ( p_drv_part2 == NULL )
         {
              handle_error();
         }
         else
         {
              ...
              p_drv_usr->getstatus( p_drv_usr );
              ...
              p_drv_usr->getstatus( p_drv_part2 );
              ...
         }
    }
```

To initialize the file system volume on Partition3 of eMMC card0 use the code as shown above, but substitute the following:

```
...
driver_param = MMCSD_BUILD_PARAMETER( 0u, EMMC_PARTITION_IDX_PARTITION_3 );
ret = f_initvolume( 0, mmcsd_initfunc, driver_param );
...
```

Card Health-related Information

These elements control the information that may be returned by the **emmc_get_health_status()** function:

Element	Information returned (detailed below this table)
EMMC_PRE_EOL_INFO	An indication of device lifetime, based on the average number of reserved blocks.
EMMC_DEVICE_LIFE_TIME_EST_A	An estimated indication of the device lifetime, based on the averaged wear-out of memory of Type A (SLC) relative to its maximum estimated device lifetime.
EMMC_DEVICE_LIFE_TIME_EST_B	An estimated indication of the device lifetime, based on the averaged wear-out of memory of Type B (MLC) relative to its maximum estimated device lifetime.
EMMC_VENDOR_PROP_HEALTH_REPORT	A vendor proprietary card health report. This is vendor-dependent. The register content is written to a user buffer in raw format.

EMMC_PRE_EOL_INFO

This gives an indication of device lifetime, reflected by average reserved blocks.

Element	Description
EMMC_PRE_EOL_INFO_NOT_DEFINED	Not defined, in other words not supported by the card.
EMMC_PRE_EOL_INFO_NORMAL	The status is normal.
EMMC_PRE_EOL_INFO_WARNING	Warning: 80% of reserved blocks have been consumed.
EMMC_PRE_EOL_INFO_URGENT	The condition of the device is serious.

EMMC_DEVICE_LIFE_TIME_EST_A and EMMC_DEVICE_LIFE_TIME_EST_B

Use these to select life time estimation (indications of device lifetime reflected by the average reserved blocks of Type A/B). Possible outputs are:

Element	Description
EMMC_DEVICE_LIFE_TIME_EST_0_10	0% - 10% of the device lifetime has been used.
EMMC_DEVICE_LIFE_TIME_EST_10_20	10% - 20% of the device lifetime has been used.
EMMC_DEVICE_LIFE_TIME_EST_20_30	20% - 30% of the device lifetime has been used.
EMMC_DEVICE_LIFE_TIME_EST_30_40	30% - 40% of the device lifetime has been used.
EMMC_DEVICE_LIFE_TIME_EST_40_50	40% - 50% of the device lifetime has been used.
EMMC_DEVICE_LIFE_TIME_EST_40_50	50% - 60% of the device lifetime has been used.
EMMC_DEVICE_LIFE_TIME_EST_60_70	60% - 70% of the device lifetime has been used.
EMMC_DEVICE_LIFE_TIME_EST_70_80	70% - 80% of the device lifetime has been used.
EMMC_DEVICE_LIFE_TIME_EST_80_90	80% - 90% of the device lifetime has been used.
EMMC_DEVICE_LIFE_TIME_EST_90_100	90% - 100% of the device lifetime has been used.
EMMC_DEVICE_LIFE_TIME_EST_EXCEEDED	The device has exceeded its maximum estimated device lifetime.

t_emmc_bkops_mode

This structure is used for the background operations MMCSD_IOCTL_BKOPS_GET_MODE and MMCSD_IOCTL_BKOPS_SET_MODE IOCTL calls.

- MMCSD_IOCTL_BKOPS_GET_MODE fills *b_is_supported*, *b_auto_en*, and *b_manual_en*.
- MMCSD_IOCTL_BKOPS_SET_MODE ignores *b_is_supported* and uses *b_auto_en* and *b_manual_en* for changing mode.

Element	Type	Description
b_is_supported	uint8_t	Set TRUE if background operations are supported by the eMMC.
b_auto_en	uint8_t	Set TRUE if automatic operations are supported by the eMMC.
b_manual_en	uint8_t	Set TRUE to enable manual operations (so MMCSD_IOCTL_BKOPS_START can be used). **This bit is one time programmable. It cannot be cleared after it is enabled.**

5. Integration

This section specifies the single element of this package that needs porting, depending on the target environment.

5.1. PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer.

The module makes use of the following standard PSP macro:

Macro	Package	Element	Description
PSP_RD_LE24	psp_base	psp_endianness	Reads a 24 bit value stored as little-endian from a memory location.

6. Example Code

This example shows how to use the partitioning functions to partition a Toshiba THGBMAG5A1JBAIR eMMC card. Follow each section below in turn.

Obtaining Partitioning Settings

Use the following code to get partitioning settings before a card is partitioned (typically when using the card for the first time):

```
t_emmc_partition_settings emmc_part_settings;
ret = emmc_get_partition_config( 0u, &emmc_part_settings );
```

The *emmc_part_settings* at this point show the following:

- *b_cpsi_part_supp* = TRUE, meaning that partitioning is supported.
- *b_cpsi_enh_supp* = TRUE, meaning that enhanced mode is supported.
- *b_cpsi_rel_wr_supp* = TRUE, meaning that reliable write is supported.
- *b_cpsi_part_complt* = FALSE, meaning that partitioning is not completed.

As partitioning is not completed, the whole area to be used by the user is the User Data Area. Look for its size:

```
cpsi_user_dsc.udsc_size = 966656
```

To get the real size from this, you need the multiplication factor *cpsi_mul_factor*:

```
cpsi_mul_factor = 8
```

Thus, the full device sector number is:

cpsi_user_dsc.udsc_size * *cpsi_mul_factor* = 966656 * 8 = 7733248

(The device size is 7733248 * 512B = 3959422976 = 3.6875GB.)

The maximum enhanced size is *cpsi_max_enh_size*, which is 483328. Use this together with *cpsi_mul_factor*, so it can be seen that this is exactly the half of the whole card area (*cpsi_user_dsc.udsc_size* = 966656/2 = 483328). This is the expected value for SLC vs MLC storage sizes.

Partitioning the Device

Once you have used **emmc_get_partition_config()** to get the multiplication factor, you can fill the partition sizes accordingly, as shown below:

```
t_emmc_partition_setup    emmc_part_setup;
/* User Data Area is not enhanced; does not use reliable write */
emmc_part_setup.b_cpsu_user_enh = FALSE;
emmc_part_setup.b_cpsu_user_rel_wr = FALSE;
/* 65536*8*512B = 256MB partition. */
emmc_part_setup.cpsu_part_dsc[0].pdsc_size = 65536u;
emmc_part_setup.cpsu_part_dsc[0].b_pdsc_enh = FALSE;
emmc_part_setup.cpsu_part_dsc[0].b_pdsc_rel_wr = FALSE;
/* 131072*8*512B = 512B partition. Enhanced mode will be enabled so it occupies twice
this size: 1GB */
emmc_part_setup.cpsu_part_dsc[1].pdsc_size = 131072u;
emmc_part_setup.cpsu_part_dsc[1].b_pdsc_enh = TRUE;
emmc_part_setup.cpsu_part_dsc[1].b_pdsc_rel_wr = FALSE;
/* 262144*8*512B = 1GB partition. Enhanced mode will be enabled so it occupies twice
this size: 2GB */
emmc_part_setup.cpsu_part_dsc[2].pdsc_size = 262144u;
emmc_part_setup.cpsu_part_dsc[2].b_pdsc_enh = TRUE;
emmc_part_setup.cpsu_part_dsc[2].b_pdsc_rel_wr = TRUE;
/* 32768*8*512B = 128MB partition. */
emmc_part_setup.cpsu_part_dsc[3].pdsc_size = 32768u;   /* 128MB */
emmc_part_setup.cpsu_part_dsc[3].b_pdsc_enh = FALSE;
emmc_part_setup.cpsu_part_dsc[3].b_pdsc_rel_wr = TRUE;
/* User Data Area remains: 3776MB - ( 256M + 2*512M + 2*1G + 128M ) =*/
/* = 3776MB - 3456MB = 320MB (this is what we expect) */

ret = emmc_set_partition_config( 0u, &emmc_part_setup );
```

Restarting and Getting Partition Configuration from the Partitioned Device

```
t_emmc_partition_settings emmc_part_settings;
ret = emmc_get_partition_config( 0u, &emmc_part_settings );
```

The *emmc_part_settings* now show the following:

- *b_cpsi_part_supp* = TRUE, meaning that partitioning is supported.
- *b_cpsi_enh_supp* = TRUE, meaning that enhanced mode is supported.
- *b_cpsi_rel_wr_supp* = TRUE, meaning that reliable write is supported.
- *b_cpsi_part_complt* = TRUE, meaning that partitioning is completed.

Since partitioning is completed, the information for the partitions is now valid:

```
cpsi_part_dsc[0].pdsc_size = 65536
cpsi_part_dsc[0].b_pdsc_enh = FALSE
cpsi_part_dsc[0].b_pdsc_rel_wr = FALSE
cpsi_part_dsc[1].pdsc_size = 131072
cpsi_part_dsc[1].b_pdsc_enh = TRUE
cpsi_part_dsc[1].b_pdsc_rel_wr = FALSE
cpsi_part_dsc[2].pdsc_size = 262144
cpsi_part_dsc[2].b_pdsc_enh = TRUE
cpsi_part_dsc[2].b_pdsc_rel_wr = TRUE
cpsi_part_dsc[3].pdsc_size = 32768
cpsi_part_dsc[3].b_pdsc_enh = FALSE
cpsi_part_dsc[3].b_pdsc_rel_wr = TRUE
cpsi_user_dsc.udsc_size = 81920
cpsi_user_dsc.b_udsc_enh = FALSE
cpsi_user_dsc.b_udsc_rel_wr = FALSE
```

The User Data Area size is 81920 * 8 * 512B = 320MB, which is the expected size.

7. Version

Version 1.10

For use with eMMC Management Driver versions 1.05 and above