SD Specifications

Part A5

SD Extensions API Simplified Specification

Version 1.00

April 10, 2017

Technical Committee

SD Card Association

Revision History

Date	Version	Changes compared to previous issue
April 10, 2017	1.00	The first release of the SD Extensions API Simplified Specification

To the extent this proposed specification, which is being submitted for review under the IP Policy, implements, incorporates by reference or refers to any portion of versions 1.0 or 1.01 of the SD Specifications (including Parts 1 through 4), adoption of the proposed specification shall require Members utilizing the adopted specification to obtain the appropriate licenses from the SD-3C, LLC, as required for the utilization of those portion(s) of versions 1.0 or 1.01 of the SD Specifications.
For example, implementation of the SD Specifications in a host device under versions 1.0 or 1.01 and under the adopted specification requires the execution of a SD Host Ancillary License Agreement with the SD-3C, LLC; and implementation of the SD Specifications under versions 1.0 or 1.01 and under the proposed specification in a SD Card containing any memory storage capability (other than for storage of executable code for a controller or microprocessor within the SD Card) requires the execution of a SD Memory Card License Agreement with the SD-3C, LLC.

Release of SD Simplified Specification/Addendum

The following conditions apply to the release of the SD Simplified Specification/Addendum by the SD Card Association. The Simplified Specification/Addendum is a subset of the complete version of SD Specification/Addendum that is owned by the SD Card Association.

Conditions for publication

Publisher:

SD Card Association
2400 Camino Ramon, Suite 375
San Ramon, CA 94583 USA
Telephone: +1 (925) 275-6615,
Fax: +1 (925) 886-4870
E-mail: office@sdcard.org

Notes:

This Simplified Specification is provided on a non-confidential basis subject to the disclaimers below. Any implementation of the Simplified Specification may require a license from the SD Card Association or other third parties.

Disclaimers:

The information contained in the Simplified Specification is presented only as a standard specification for SD Cards and SD Host/Ancillary products and is provided "AS-IS" without any representations or warranties of any kind. No responsibility is assumed by the SD Card Association for any damages, any infringements of patents or other right of the SD Card Association or any third parties, which may result from its use. No license is granted by implication, estoppel or otherwise under any patent or other rights of the SD Card Association or any third party. Nothing herein shall be construed as an obligation by the SD Card Association to disclose or distribute any technical information, know-how or other confidential information to any third party.

Conventions Used in This Document

Naming Conventions

- Some terms are capitalized to distinguish their definition from their common English meaning.
- Words not capitalized retain their common English meaning.

Numbers and Number Bases

- Hexadecimal numbers are written with a lower case "h" suffix or zero plus a lower case alphabet “0x” prefix, e.g., FFFFh, 80h, 0xFFFF and 0x80.
- Binary numbers are written with a lower case "b" suffix (e.g., 10b).
- Binary numbers larger than four digits are written with a space dividing each group of four digits, as in 1000 0101 0010b.
- All other numbers are decimal.

Byte and Bit Order

- All data are stored in most significant bit first
- Endian of data stored in USHORT, UINT or ULONG follows system OS’s rule.
- The data stored in BYTE pointer or the other type than USHORT, UINT and ULONG are in most significant byte.

Key Words

- May: Indicates flexibility of choice with no implied recommendation or requirement.
- Shall: Indicates a mandatory requirement. Designers shall implement such mandatory requirements to ensure interchangeability and to claim conformance with the specification.
- Should: Indicates a strong recommendation but not a mandatory requirement. Designers should give strong consideration to such recommendations, but there is still a choice in implementation.

Application Notes

Some sections of this document provide guidance to the host implementers as follows:

Application Note: This is an example of an application note.

Table of Contents

1. Overview 1

2. Block Diagram 2
2.1 SD Extension Manager and SD Device Manager 2

3. Functionality 3

4. System Requirements 4
4.1 OS 4
4.2 Format 4

5. List of Functions 5

6. Details of Functions 7
6.1 SDEM Initialization/Termination 8
6.1.1 SDSysInit 8
6.1.2 SDSysFini 9
6.2 SDDM Initialization/Termination 10
6.2.1 SDInit 10
6.2.2 SDFini 11
6.3 Drive Map 12
6.3.1 SDSysDrive 12
6.3.2 SDEnumSDDrive 13
6.4 SD Extension Info 14
6.4.1 SDGetVersion 14
6.4.2 SDGetCapability 15
6.5 Register Access 17
6.5.1 SDGetCSD 17
6.5.2 SDGetCID 18
6.5.3 SDGetSDStatus 19
6.5.4 SDGetSCR 20
6.5.5 SDGetOCR 21
6.6 Extension Register Access 22
6.6.1 SDReadExSingle 22
6.6.2 SDReadExMulti 23
6.6.3 SDWriteExSingle 24
6.6.4 SDWriteExMulti 25
6.6.5 SDSetCallBack 26
6.7 Erase Command 27
6.7.1 SDErase 27
6.8 Vendor Specific Command 28
6.8.1 SDGenCmd 28
6.9 Drive Lock 29
6.9.1 Lock the Drive 29
6.9.2 Unlock the Drive 30

7. List of Return Codes 31
7.1 Return Codes 31
7.2 Table of Return Codes and Functions 32
7.2.1 SD Extension Manager (SDEM) 32
7.2.2 SD Device Manager (SDDM) 33

8. File Name and Location 34
8.1 Windows 34
8.1.1 SD Extension Manager (SDEM) 34
8.1.2 SD Device Manager (SDDM) 34
8.1.3 Location 34
8.1.4 File Versioning 34

9. Usage 35

10. Additional Information 35

Appendix A (Normative) : Reference 36
A.1 Reference 36

Appendix B (Normative) : Special Terms 37
B.1 Terminology 37
B.2 Abbreviations 37
B.3 Expression for function 37

Table of Figures

Figure 8-1 : File name manner of DLL for Device Manager 34

Table of Tables

Table 5-1 : SD Extensions API List 5
Table 5-2 : Driver's Type and Function Support 6
Table 6-1 : API's Return Code at Locked Drive 29
Table 7-1 : Return Codes List 31
Table 7-2 : Return Codes of SD Extension Manager (SDEM) 32
Table 7-3 : Return Codes of SD Device Manager (SDDM) 33

1. Overview

SD Memory Card supports some extended features other than memory read/write. This SD Extensions API Specification provides a standardized set of the API that is essential to access such extended features for SD Memory Card. A standardized set of the API is beneficial for various platform environments including Personal Computer (PC) or smart phone environment.

This SD Extensions API Specification specifies two layers of modules which are SD Extension Manager (SDEM for short) and SD Device Manager (SDDM for short). This SD Extensions API Specification contains implementation notes for those two modules.

The SDEM provides a common interface for interacting with multiple applications and underlying attached multiple SDDMs.

The SDDM provides an interface for interacting with SDEM and underlying SD Host Controller hardware directly or via a driver to control the SD Host Controller hardware. The driver may access SD Host Controller hardware via USB bus or ATA bus

2. Block Diagram

2.1 SD Extension Manager and SD Device Manager

There are some application, one SDEM, some SDDM, some device drivers, some SD Host Controllers and some SD Memory Cards. An application in the figure utilizes an extension feature through SDEM and SDDM.

Following of Chapter 2 is a blank in the simplified specification.

3. Functionality

SD Extension System provides the following functions.

- **System Initialization/Termination**
  It provides a function of initialization and termination of the SD Extension System. SDEM initializes in this process. Memory to control handles given by SDDMs is allocated in the initialization process. The allocated memory is released in the termination process.

- **SDDM Initialization/Termination**
  It provides a function of creating a handle for an application to access an SD Memory Card using an SDDM. This handle is used as an identifier to access the SD Memory Card. The handle is released in the termination process.

- **Drive Map**
  It provides mapping information between an SD Memory Card which is accessible by the SD Extension API and a drive letter.

- **SD Extension Info**
  It provides information of SDEM and SDDMs, such as version number and capability.

- **Register Access**
  It provides a function of access to registers of the SD Memory Card.

- **Extension Register Access**
  It provides a function of access to Extension Register of the SD Memory Card.

- **Erase**
  It provides a function to issue SD commands for erase.

- **Vendor Specific Command**
  It provides a function to issue a vendor specific command of SD Memory Card.

- **Drive Lock**
  An application can exclusively use this API to a specific drive.

4. System Requirements

4.1 OS

SDEM and SDDM and shared objects for Windows 10 operation system.

4.2 Format

SDEM and SDDM are dynamic-link library (.dll) formats and operated in the user mode.

5. List of Functions

SD Extensions API Software support functions described below:

Table 5-1 : SD Extensions API List

Function Name	API Name	SD CMD	Details
System Initialization/ Termination	SDSysInit	N/A	Initialize system
	SDSysFini	N/A	Terminate system
SDDM Initialization/ Termination	SDInit	N/A	Initialize driver software
	SDFini	N/A	Terminate driver software
Drive Map	SDSysDrive	N/A	Map a unique id number with the Driver Letter (Only for SDDM)
	SDEnumSDDrive	N/A	Enumerate the SD drives (Only for SDEM)
SD Extension Info	SDGetVersion	N/A	Acquires SDEM and SDDM version
	SDGetCapability	N/A	Acquires SDEM and SDDM capability
Register Access	SDGetCSD	CMD9	Retrieves the CSD(Card Specific Data) register
	SDGetCID	CMD10	Retrieves the CID (Card Identification Number) register
	SDGetSDStatus	ACMD13	Retrieves the SD Status
	SDGetSCR	ACMD51	Retrieves the SCR register
	SDGetOCR	ACMD41	Retrieves the OCR(Operation Condition Register) register
Extension Register Access	SDReadExSingle	CMD48	Read a single block of Extension Register
	SDReadExMulti	CMD58	Read multiple blocks of Extension Register
	SDWriteExSingle	CMD49	Write a single block of Extension Register
	SDWriteExMulti	CMD59	Write multiple blocks of Extension Register
	SDSetCallBack	N/A	Set call back function for event detection
Erase	SDErase	CMD32, CMD33, CMD38	Erase data in user area
Vendor Specific Command	SDGenCmd	CMD56	Vendor Specific Command
Drive Lock	SDLockDrive	N/A	Lock the drive
	SDUnlockDrive	N/A	Unlock the drive

Some API may not be supported. The Table 5-2 below shows mandatory or option of API. M means mandatory. O means optional. In case of option, the API may be implemented by the SDDM or SDEM.

Table 5-2 : Driver's Type and Function Support

API groups	Related APIs	Application SDEM	Application SDDM	Common SDEM	Common SDDM
Basic API set	SDSysInit, SDSysFini, SDInit, SDFini, SDEnumSDDrive (only for SDEM), SDSysDrive (only for SDDM), SDGetVersion and SDGetCapability.	M	M	M	M
Register Access	SDGetCSD, SDGetCID, SDGetSDStatus, SDGetSCR and SDGetOCR	O	O	M	M
Extension Register Access	SDReadExSingle, SDReadExMulti, SDWriteExSingle and SDWriteExMulti	O	O	M	M
Erase	SDErase	O	O	M	M
Drive Lock	SDLockDrive and SDUnlockDrive	O	O	M	M
Vendor Specific Command	SDGenCmd	O	O	M	M
Event Detection	SDSetCallBack	O	O	M	O

6. Details of Functions

The descriptive format for each interface is as follows.

x.x.x SDFunctionName

Function number and function name.

[Format]

```
UINT   SDFunctionName (UINT param);
```

Describes the name of the interface.

[Argument]

[IN] `UINT` `param` Description of the arguments.

Describes the arguments.
If an error code returns, application or SDEM shall treat data in argument as invalid unless otherwise specified.

[Return Value]

0: Success, Positive number: Error code

Describes return value.

[Description]

Describes an act of this interface and a notice. It is for application developer, SDEM and SDDM.

[Remarks]

Describes an additional information. It is for application developer, SDEM and SDDM.

[Implementation Note for SDEM]

Describes the implementation note for SDEM and aim at

- standardizing the action of SDEM,
- description for how SDEM uses SDDM for SDDM developer and
- description for how SDEM act on for application developer.

The behavior of SDEM should be in accordance with this implementation note.

[Implementation Note for SDDM]

Describes implementation note for SDDM and aim at

- standardizing the action of SDDM and
- description for how SDDM uses device driver for SDDM developer.

The behavior of SDDM should be in accordance with this implementation note.

6.1 SDEM Initialization/Termination

6.1.1 SDSysInit

[Format]

```
UINT   SDSysInit(void);
```

[Argument]
None

[Return Value]
0: Success, Positive number: Error code

[Description]
This function initializes the SD Extension System.
It initializes a handle management buffer.

[Remarks]
Call this function once to initialize the system.
A return value first called by each process of caller is SD_E_SUCCESS.

[Implementation Note for SDEM]
This interface is succeeded by the first call on a process; it is not relation to the other process.
Enumerate the all SDDMs which the SDEM can access and load them.

[Implementation Note for SDDM]
This interface is succeeded by the first call on a process; it is not relation to the other process.

6.1.2 SDSysFini

[Format]

```
UINT   SDSysFini(void);
```

[Argument]
None

[Return Value]
0: Success, Positive number: Error code

[Description]
This function terminates the system.

[Remarks]
Call this function once to terminate the system.
Before calling this function, the application should call SDFini, return and close all handles given by SDInit. Otherwise, SDEM and SDDM return SD_E_HANDLE_OPENED error.

[Implementation Note for SDEM]
SDEM checks whether or not all handles by the SDEM are closed. If a handle opened by SDInit is still opened, SDEM shall return SD_E_HANDLE_OPENED. After check, SDEM unloads the all SDDMs which are loaded by SDSysInit and returns SD_E_SUCCESS.
This is succeeded by the first call on a process; it is not relation to the other process.

[Implementation Note for SDDM]
SDDM checks whether or not all handles by the SDDM are closed. If a handle opened by SDInit is still opened, SDDM shall return SD_E_HANDLE_OPENED. This interface is succeeded by the first call on a process; it is not relation to the other process.

6.2 SDDM Initialization/Termination

6.2.1 SDInit

[Format]

```
UINT   SDInit(UINT *handle, USHORT Drive);
```

[Argument]

[OUT]	UINT	*handle	Handle
[IN]	USHORT	Drive	Drive Letter (A=1, B=2, ...)

[Return Value]

0: Success, Positive number: Error code

[Description]

This function initializes the driver software.
It stores a handle.

[Remarks]

An application or function middleware accesses the SD Memory Card using the handle acquired by this function.
The handle acquired by this function must be returned by the SDFini function without fail. If SDInit results in an error, a *handle value will be undefined and it will not be necessary to release the handle by using SDFini.

[Implementation Note for SDEM]

SDEM calls SDInit of the SDDM which treat the SD Memory Card specified by the Drive Letter. SDEM can check if the drive is supported or not by SDSysDrive interface to all SDDM that is already loaded. When the SDDM returns SD_E_SUCCESS with a handle, SDEM returns SD_E_SUCCESS with a handle which issued by SDEM. The value of handle shall be unique among handles provided by the SDEM.

[Implementation Note for SDDM]

Enumerate SD drives which this SDDM can support, and check if the drive number pass from SDEM is included or not. If the drive number pass from SDEM is included in supported drive, this interface should be succeeded, and initialize this drive at that time. The value of handle shall be unique among handles provided the SDDM.

6.2.2 SDFini

[Format]

```
UINT   SDFini(UINT handle);
```

[Argument]

[IN] `UINT` `handle` Handle

[Return Value]

0: Success, Positive number: Error code

[Description]

This function terminates the driver software.

[Remarks]

The handle acquired by the SDInit function must be returned by this function without fail.

[Implementation Note for SDEM]

None.

[Implementation Note for SDDM]

If the handle is valid value, that means called SDInit are succeeded, then close the handle.

6.3 Drive Map

6.3.1 SDSysDrive

[Format]

```
UINT   SDSysDrive(USHORT id, USHORT *pDrive);
```

[Argument]

[IN]	USHORT	Id	id number
[OUT]	USHORT	*pDrive	Drive Letter (A=1, B=2, …)

[Return Value]

0: Success, Positive number: Error code

[Description]

This function acquires the drive letter which SDDM manages.
Each SDDM uses the id number to uniquely manage the SD Memory Card device. The id number starts with 0, and ends with 26 or less.
The id number is not necessarily a consecutive number.
If an unassigned id number is specified, it returns SD_E_ID_INVALID.
If the specified id number is greater than the number managed by the driver, it returns SD_E_ID_OVERFLOW.
If the specified id number is assigned as the SD Memory Card drive, the drive letter is stored in pDrive and it returns SD_E_SUCCESS.
Note: If any drive is not allocated since it is not formatted, 0 is stored in pDrive and it returns SD_E_SUCCESS. By using this, SDEM can omit the search processing which is not useful.

[Remarks]

This function is implemented only in each SDDM.

[Implementation Note for SDEM]

None.

[Implementation Note for SDDM]

Enumerate the SD drive that is supported by this SDDM. Id number can be assigned freely between 0 and 26 or less for the SD drive which SDEM supports.
If the assigned id number is specified, return SD_E_SUCCESS with pDrive(Drive Letter information). If the unassigned id number is specified, return SD_E_ID_INVALID with pDrive =0. But, if the id number which is not formatted is specified, return SD_E_SUCCESS whit pDrive=0.
If the id number which is specified is greater than the number managed by the SDDM, it returns SD_E_ID_OVERFLOW.
This interface returns SD supported drive. SD supported drive means that the drive can be SD drive.
Ex) There is the drive which can be inserted SD or MMC, if the SD Memory Card is not inserted or MMC is inserted, the drive is SD supported drive.

6.3.2 SDEnumSDDrive

[Format]

```
UINT   SDEnumSDDrive(UINT *pSDDrive, void *pReserved);
```

[Argument]

[OUT]	UINT	*pSDDrive	SD drive status
[OUT]	void	*pReserved	Used for expansion

[Return Value]

0: Success, Positive number: Error code

[Description]

This function enumerates SD Memory Card device drives.
Each bit for *pSDDrive is set as follows. Bit 1 to 26 in the Drive Map Information each correspond to drive letters “A” to “Z”.

Name	Width	slice	Value
Reserved	5	[31: 27]	Always 0
Drive Map Information	26	[26: 1]	0: non-SD drive or drive does not exist. 1: SD drive
Reserved	1	[0: 0]	Always 0

*pReserved is a variable for future expansion. This version specifies NULL.

[Remarks]

This function is implemented only in SDEM.
This function enumerates drives that can be the SD drive. Thus, a Drive Map Information value shall be “1” even in the case that it is an SD drive but an SD Memory Card is not inserted.

[Implementation Note for SDEM]

To check the drive that can support SD or not, call SDSysDrive interface to all loaded SDEM. Then set the result to the buffer located by pSDDrive pointer.
Set bit ‘1’ for SD supported drive and set ‘0’ for un-supported drive. SD supported drive means that the drive can be SD drive. Example: There is the drive which can be inserted SD, if the SD Memory Card is not inserted; the drive is SD supported drive.

[Implementation Note for SDDM]

None.

6.4 SD Extension Info

6.4.1 SDGetVersion

[Format]

```
UINT   SDGetVersion(USHORT  *SDEMVersion,  USHORT  *SDDMVersion,  UINT
handle);
```

[Argument]

[OUT]	USHORT	*SDEMVersion	Pointer to version number of SDEM
[OUT]	USHORT	*SDDMVersion	Pointer to version number of SDDM
[IN]	UINT	Handle	Handle

[Return Value]

0: Success, Positive number: Error code

[Description]

It provides a version number of the SDEM and SDDM specified by the handle.
In SDEMVersion, one of the following numbers is set.
10h: SDEM complies with the SD Extensions API Specification version 1.00
Others: Reserved
In SDDMVersion, one of the following numbers is set.
10h: SDDM complies with the SD Extensions API Specification version 1.00
Others: Reserved

[Remarks]

Application can call this API before getting a handle. In this case, application can get the SDEMVersion only.

[Implementation Note for SDEM]

SDEM sets SDEMVersion properly. SDEM calls SDGetVersion of the designated SDDM to get SDDMVersion. If the handle is not assigned yet, SDEM sets nothing to SDDMVersion and returns SD_E_SUCCESS.

[Implementation Note for SDDM]

SDDM sets SDDMVersion properly. SDEMVersion is set to 00h.

6.4.2 SDGetCapability

[Format]

```
UINT   SDGetCapability(UINT   *SDEMCapability,   UINT   *SDDMCapability,
UINT handle);
```

[Argument]

[OUT]	BYTE	*SDEMCapability	Pointer to SDEM capability
[OUT]	BYTE	*SDDMCapability	Pointer to SDDM capability
[IN]	UINT	Handle	Handle

[Return Value]

0: Success, Positive number: Error code

[Description]

It provides a capability of the SDEM and SDDM specified by the handle.
SDEMCapability or SDDMCapability is 256-bits data and shows capability of SDEM or SDDM. The meaning is following.

Field name	Width	Slice	Related APIs	Meaning
Header	16	[255:240]	None	"SD" in ASCII
Register Access	1	239	SDGetCSD, SDGetCID, SDGetSDStatus, SDGetSCR and SDGetOCR	0: Not supported 1: Supported
Extension Register Access	1	238	SDReadExSingle, SDReadExMulti, SDWriteExSingle and SDWriteExMulti	0: Not supported 1: Supported
Erase	1	237	SDErase	0: Not supported 1: Supported
Drive Lock	1	236	SDLockDrive and SDUnlockDrive	0: Not supported 1: Supported
Vendor Specific Command	1	235	SDGenCmd	0: Not supported 1: Supported
Reserved	11	[234:224]	None	All bits shall be set to 0
Event Detection	8	[223:216]	SDSetCallBack	0x00: Not supported 0x01: Periodical read of FXE Register Set supported (Only for SDDM) 0x02: FX_EVENT bit detection supported (Only for SDDM) 0xFF: Supported (Only for SDEM) Others: reserved
Reserved		[215:0]		All bits shall be set to 0

If a field is set to “not supported,” the SDDM or SDEM does not have the related APIs.

[Remarks]

Application can call this API before getting a handle. In this case, application can get the SDEM capability only.

[Implementation Note for SDEM]

SDEM sets SDEM capability properly. SDEM calls SDGetCapability of the designated SDDM to get SDDM capability. If the handle is not assigned yet, SDEM sets nothing to SDDMCapability and returns SD_E_SUCCESS.

[Implementation Note for SDDM]

SDDM sets SDDMCapability properly. SDEMCapability is set to all 0.

6.5 Register Access

6.5.1 SDGetCSD

[Format]

```
UINT   SDGetCSD(BYTE *CSDRegister, UINT handle);
```

[Argument]

[OUT]	BYTE	*CSDRegister	CSD register
[IN]	UINT	handle	Handle

[Return Value]

0: Success, Positive number: Error code

[Description]

This function retrieves the CSD (Card Specific Data) register (CSD Version 1.0 or CSD Version 2.0) of the SD Memory Card in the drive which is associated with the handle. The format of CSD registers are described in Section 5.3.2 and Section 5.3.3 of the Part 1 Physical Layer Specification.

[Remarks]

Application developers: The allocated size of CSD Register shall be 16 bytes.

[Implementation Note for SDEM]

SDEM calls SDGetCSD of the SDDM with the handle.

[Implementation Note for SDDM]

The CSD register can be retrieved by using CMD9. SDDM sets the retrieved data of CSD Register to the *CSDRegister of this API.
When the API is being called with a handle and the drive has been already locked by SDLockDrive with another handle, SDDM shall return SD_E_DRIVE_LOCKED.

6.5.2 SDGetCID

[Format]

```
UINT   SDGetCID(BYTE *CIDregister, UINT handle);
```

[Argument]

[OUT]	BYTE	*CIDRegister	CID register
[IN]	UINT	handle	Handle

[Return Value]

0: Success, Positive number: Error code

[Description]

This function retrieves the CID (Card IDentification) register of the SD Memory Card in the drive which is associated with the handle. The format of CID register is described in Section 5.2 of the Part 1 Physical Layer Specification.

[Remarks]

Application developers: The allocated size of CID register shall be 16 bytes.

[Implementation Note for SDEM]

SDEM calls SDGetCID of the SDDM with the handle.

[Implementation Note for SDDM]

The CID register can be retrieved by using CMD10. SDDM sets the retrieved data of CID register to the *CIDregister of this API.

When the API is being called with a handle and the drive has been already locked by SDLockDrive with another handle, SDDM shall return SD_E_DRIVE_LOCKED.

6.5.3 SDGetSDStatus

[Format]

```
UINT   SDGetSDStatus(BYTE *SDStatus, UINT handle);
```

[Argument]

[OUT]	BYTE	*SDStatus	SD Status
[IN]	UINT	handle	Handle

[Return Value]

0: Success, Positive number: Error code

[Description]

This function retrieves the SD Status of the SD Memory Card in the drive which is associated with the handle. The format of SD Status is described in Section 4.10.2 of the Part 1 Physical Layer Specification.

[Remarks]

Application developers: The allocated size of SD Status shall be 64 bytes.

[Implementation Note for SDEM]

SDEM calls SDGetSDStatus of the SDDM with the handle.

[Implementation Note for SDDM]

The SD Status can be retrieved by using ACMD13. SDDM sets the retrieved data of SD Status to the *SDStatus of this API.

When the API is being called with a handle and the drive has been already locked by SDLockDrive with another handle, SDDM shall return SD_E_DRIVE_LOCKED.

6.5.4 SDGetSCR

[Format]

```
UINT   SDGetSCR(BYTE *SCRregister, UINT handle);
```

[Argument]

[OUT]	BYTE	*SCRRegister	SCR register
[IN]	UINT	handle	Handle

[Return Value]

0: Success, Positive number: Error code

[Description]

This function retrieves the SCR (SD Card Configuration Register) register of the SD Memory Card in the drive which is associated with the handle. The format of SCR register is described in Section 5.6 of the Part 1 Physical Layer Specification.

[Remarks]

Application developers: The allocated size of SCR register shall be 8 bytes.

[Implementation Note for SDEM]

SDEM calls SDGetSCR of the SDDM with the handle.

[Implementation Note for SDDM]

The SCR register can be retrieved by using ACMD51. SDDM sets the retrieved data of SCR register to the *SCRregister of this API.
When the API is being called with a handle and the drive has been already locked by SDLockDrive with another handle, SDDM shall return SD_E_DRIVE_LOCKED.

6.5.5 SDGetOCR

[Format]

```
UINT   SDGetOCR(BYTE *OCRregister, UINT handle);
```

[Argument]

[OUT]	BYTE	*OCRRegister	OCR register
[IN]	UINT	handle	Handle

[Return Value]

0: Success, Positive number: Error code

[Description]

This function retrieves the OCR (Operation Condition Register) register of the SD Memory Card in the drive which is associated with the handle. The format of OCR register is described in Section 5.1 of the Part 1 Physical Layer Specification.

[Remarks]

Application developers: The allocated size of OCR register shall be 4 bytes.

[Implementation Note for SDEM]

SDEM calls SDGetOCR of the SDDM with the handle.

[Implementation Note for SDDM]

The OCR register can be retrieved by using ACMD41. SDDM sets the retrieved data of OCR register to the `OCRregister of this API.

When the API is being called with a handle and the drive has been already locked by SDLockDrive with another handle, SDDM shall return SD_E_DRIVE_LOCKED.

6.6 Extension Register Access

6.6.1 SDReadExSingle

[Format]

```
UINT   SDReadExSingle(BYTE *buf, ULONG *length, ULONG address,
                                  BYTE mio, BYTE fno, UINT handle);
```

[Argument]

[OUT]	BYTE	*buf	Buffer for read data
[IN/OUT]	ULONG	*length	Read size in bytes
[IN]	ULONG	address	Address of extension register space
[IN]	BYTE	mio	Selection of memory space or IO space mio = 0x00 : Memory Extension mio = 0x01 : I/O Extension
[IN]	BYTE	fno	Function number
[IN]	UINT	handle	Handle

[Return Value]

0: Success, Positive number: Error code

[Description]

This function reads data from the Extension Register of SD Memory Card using CMD48. A pointer to the BYTE array to store data from the Extension Register is designated as first argument. An expected length in byte of data to be stored at the buffer is set in the second argument. When accessing to the Data Port, the argument length shall be zero. The argument address is an address of Extension Register Space. The mio argument selects memory space or I/O space (0x00: Memory Space Extension, 0x01: I/O Space Extension, other values are reserved for future use). The argument fno is a function number to be accessed.

[Remarks]

Application developers: The allocated size of buf shall be 512 bytes despite of the length argument.

[Implementation Note for SDEM]

SDEM calls SDReadExSingle of the SDDM with the same arguments.

[Implementation Note for SDDM]

SDDM sends the SD Memory Card CMD48 with arguments given by calling this API. The read data is set to the buffer indicated by *buf.

When the API is being called with a handle and the drive has been already locked by SDLockDrive with another handle, SDDM shall return SD_E_DRIVE_LOCKED.

6.6.2 SDReadExMulti

[Format]

```
UINT   SDReadExMulti(BYTE *buf, ULONG *length, ULONG address,
                                    BYTE mio, BYTE fno, UINT handle);
```

[Argument]

[OUT]	BYTE	*buf	Buffer for read data
[IN/OUT]	ULONG	*length	Read size in bytes
[IN]	ULONG	address	Address of extension register space
[IN]	BYTE	Mio	Selection of memory space or IO space mio = 0x00 : Memory Extension mio = 0x01 : I/O Extension
[IN]	BYTE	fno	Function number
[IN]	UINT	handle	Handle

[Return Value]

0: Success, Positive number: Error code

[Description]

This function reads data from the Extension Register of SD Memory Card using CMD58. A pointer to the BYTE array to store data from the Extension Register is designated as first argument. An expected length in byte of data to be stored at the buffer is set in the second argument. The value of the length argument shall be 512-byte (0x0200) or 32K-byte (0x8000) aligned. The argument address is an address of Extension Register Space. The mio argument selects memory space or I/O space (0x00: Memory Space Extension, 0x01: I/O Space Extension, other values are reserved for future use). The argument fno is a function number to be accessed.

[Remarks]

Application developers: The allocated size of buf shall be at least 512 bytes and shall be aligned with 512-byte (0x0200) alignment or 32K-byte (0x8000) alignment as well as the length argument.

[Implementation Note for SDEM]

SDEM calls SDReadExMulti of the SDDM with the same arguments..

[Implementation Note for SDDM]

SDDM sends the SD Memory Card CMD58 with arguments given by calling this API. The read data is set to the buffer indicated by *buf.
When the API is being called with a handle and the drive has been already locked by SDLockDrive with another handle, SDDM shall return SD_E_DRIVE_LOCKED.

6.6.3 SDWriteExSingle

[Format]

```
UINT   SDWriteExSingle(BYTE *buf, ULONG length, ULONG address,
                       BYTE mask, BYTE mio, BYTE fno, UINT handle);
```

[Argument]

[OUT]	BYTE	*buf	Buffer for write data
[IN]	ULONG	length	Write size in bytes
[IN]	ULONG	address	Address of extension register space
[IN]	BYTE	mask	Mask data for mask write operation
[IN]	BYTE	mio	Selection of memory space or IO space mio = 0x00 : Memory Extension mio = 0x01 : I/O Extension
[IN]	BYTE	fno	Function number
[IN]	UINT	handle	Handle

[Return Value]

0: Success, Positive number: Error code

[Description]

This function writes data to the Extension Register of SD Memory Card using CMD49. A pointer to the BYTE array to store data to the Extension Register is designated as first argument. A length of valid data in the buffer is set in the second argument. When accessing to the Data Port, the argument length shall be zero. The argument address is an address of Extension Register Space. The mask argument specifies mask data, if this argument is non-zero value, then the value is the mask data. The mio argument selects memory space or I/O space (0x00: Memory Space Extension, 0x01: I/O Space Extension, other values are reserved for future use). The argument fno is a function number.

[Remarks]

Application developers: The allocated size of buf shall be 512 bytes despite of length argument. The residual part of the buffer will be ignored however it is recommended that the area is cleared with zeros. When the mask argument is non-zero value, the length argument shall be 0x01.

[Implementation Note for SDEM]

SDEM calls SDWriteExSingle of the SDDM with the same arguments.

[Implementation Note for SDDM]

SDDM sends the SD Memory Card CMD49 with arguments given by calling this API.
When the API is being called with a handle and the drive has been already locked by SDLockDrive with another handle, SDDM shall return SD_E_DRIVE_LOCKED.

6.6.4 SDWriteExMulti

[Format]

```
UINT   SDWriteExMulti(BYTE *buf, ULONG length, ULONG address,
                                  BYTE mio, BYTE fno, UINT handle);
```

[Argument]

[OUT]	BYTE	*buf	Buffer for write data
[IN]	ULONG	length	Write size in bytes
[IN]	ULONG	address	Address of extension register space
[IN]	BYTE	mio	Selection of memory space or IO space mio = 0x00 : Memory Extension mio = 0x01 : I/O Extension
[IN]	BYTE	fno	Function number
[IN]	UINT	handle	Handle

[Return Value]

0: Success, Positive number: Error code

[Description]

This function writes data to the Extension Register of SD Memory Card using CMD59. A pointer to the BYTE array to store data to the Extension Register is designated as first argument. A valid length in byte of writing data at the buffer is set in the second argument. The value of the length argument shall be 512-byte (0x0200) or 32K-byte (0x8000) aligned. The argument address is an address of Extension Register Space. The mio argument selects memory space or I/O space (0x00: Memory Space Extension, 0x01: I/O Space Extension, other values are reserved for future use). The argument fno is a function number to be accessed.

[Remarks]

Application developers: The allocated size of buf shall be at least 512 bytes and shall be aligned with 512-byte (0x0200) alignment or 32K-byte (0x8000) alignment as well as the length argument.

[Implementation Note for SDEM]

SDEM calls SDWriteExMulti of the SDDM with the same arguments.

[Implementation Note for SDDM]

SDDM sends the SD Memory Card CMD59 with arguments given by calling this API.
When the API is being called with a handle and the drive has been already locked by SDLockDrive with another handle, SDDM shall return SD_E_DRIVE_LOCKED.

6.6.5 SDSetCallBack

[Format]

```
UINT   SDSetCallBack(UINT handle, BYTE fno, void (*func)(void));
```

[Argument]

[IN]	UINT	handle	Handle
[IN]	BYTE	Fno	Function number
[IN]	void	(*func)(void)	Function pointer for call back

[Return Value]

0: Success, Positive number: Error code

[Description]

Applications can set a callback function when the SDDM detects an event occurrence in the Function Extension specified by function number.

[Remarks]

Application developers:

[Implementation Note for SDEM]

SDEM calls SDSetCallBack of the SDDM with the same arguments. If the SDDM does not have this API, SDEM returns an error.

[Implementation Note for SDDM]

If the SDDM does not support this function, this API shall not be implemented. If the SDDM supports event detection, SDDM stores the function pointer. SDDM can choose one from two detection methods. One is to periodically read Function Extension Event Register Set by CMD48. The other is Function Extension Event (FX_EVENT) bit detection. Applications recognize the method capability by calling SDGetCapability and reading SDDMCapability. SDDM reads the Function Extension Event Register Set periodically or at the time that FX EVENT in R1 response is detected. When the Function Extension Event Register Set indicates event occurrence of the Function Extension specified by function number “fno” of this API, the SDDM calls back the function pre-set by this API.

When the API is being called with a handle and the drive has been already locked by SDLockDrive with another handle, SDDM shall return SD_E_DRIVE_LOCKED.

6.7 Erase Command

6.7.1 SDErase

[Format]

```
UINT   SDErase(ULONG  startaddr,  ULONG  endaddr,  BYTE  *cmdarg  UINT
handle);
```

[Argument]

[IN]	ULONG	startaddr	Offset to start erase process
[IN]	ULONG	endaddr	Offset to end erase process
[IN]	BYTE	*cmdarg	4-byte Command argument of CMD38
[IN]	UINT	handle	Handle

[Return Value]

0: Normal termination, Positive number: Error code

[Details]

This function erases the data of the User Data Area of the SD Memory Card. The start address of erase process is set as the first argument. The end address of erase process is set as the second argument. These addresses are the offset value from address 0 in the User Data Area of SD Memory Card. The 4-byte command argument of CMD38 is set as the third argument. The handle acquired by the SDInit function is designated as the second argument.

[Remarks]

When the erase of big size is specified, this function may take much time to return. Regardless SDSC, SDHC and SDXC, the startaddr and endaddr will be set to command argument of CMD32 and CMD33 without recalculation of the address although the start address and end address are in byte address for SDSC and block address for SDHC and SDXC.
It is recommended that the drive be un-mounted it before erase in order to refresh cache of file system.

[Implementation Note for SDEM]

SDEM calls SDErase of the SDDM with the same arguments.

[Implementation Note for SDDM]

SDDM issues CMD32, CMD33 and CMD38 consecutively for erase process. CMD32 is issued with setting the startaddr to the command argument. CMD33 is issued with setting the endaddr to the command argument. CMD38 is issued with setting the third argument of this function to the command argument.
When the API is being called with a handle and the drive has been already locked by SDLockDrive with another handle, SDDM shall return SD_E_DRIVE_LOCKED.

6.8 Vendor Specific Command

6.8.1 SDGenCmd

[Format]

```
UINT   SDGenCmd(BYTE *arg, UCHAR *data, UINT size, UINT handle);
```

[Argument]

[IN]	BYTE	*arg	Command argument of CMD56. The bit 0 of the first byte denotes direction of data transfer direction=1: reading data from the card direction=0: writing data from the card The rest of the bits are vendor-specific.
[IN/OUT]	UCHAR	*data	Data buffer to be read or written.
[IN]	UINT	size	Data block size of the data in bytes.
[IN]	UINT	handle	Handle

[Return Value]

0: Success, Positive number: Error code

[Description]

This function executes a vendor-specific command (CMD56.)

[Remarks]

Application developers: The data block size of data shall be the same size as BLOCK_LEN for Standard Capacity SD Memory Card, and 512 bytes for SDHC and SDXC Cards.

[Implementation Note for SDEM]

SDEM calls SDGenCmd of the SDDM with the same arguments.

[Implementation Note for SDDM]

SDDM sends the SD Memory Card CMD56 with arguments given by calling this API. When the direction bit of command argument is set to 1, the read data is set to the buffer indicated by *data. When the API is being called with a handle and the drive has been already locked by SDLockDrive with another handle, SDDM shall return SD_E_DRIVE_LOCKED.

6.9 Drive Lock

6.9.1 Lock the Drive

[Format]

```
UINT   SDLockDrive(UINT handle);
```

[Argument]

[IN] `UINT` `handle` Handle

[Return Value]

0: Success, Positive number: Error code

[Description]

This function locks a drive corresponding to a handle. If the process succeeds, the function returns SD_E_SUCCESS and prohibits access to this drive by other handles. If other handles have already locked this drive, the function returns SD_E_LOCK_FAILURE.
Table 6-1 describes API's return code when the drive is locked by another handle.

Table 6-1 : API's Return Code at Locked Drive

API	Return code when locked by another handle
SDSysInit	SD_SUCCESS if the process succeeds
SDSysFini	SD_SUCCESS if the process succeeds
SDInit	SD_SUCCESS if the process succeeds
SDFini	SD_SUCCESS if the process succeeds
SDEnumSDDrive (only for SDEM)	SD_SUCCESS if the process succeeds
SDSysDrive (only for SDDM)	SD_SUCCESS if the process succeeds
SDGetVersion	SD_SUCCESS if the process succeeds
SDGetCapability	SD_SUCCESS if the process succeeds
SDLockDrive	SD_E_LOCK_FAILURE
SDUnlockDrive	SD_E_UNLOCK_FAILURE
APIs other than above	SD_E_DRIVE_LOCKED

[Remarks]

Before SDFini releases the handle, the drive locked by this function is required to be unlocked by SDUnlockDrive. Access to the card without using SDEM (e.g. file copies by Explorer) is allowed.

[Implementation Note for SDEM]

If there is not any SDEM error factor, SDEM behaves like below,
If SDDM has SDLockDrive interface, call it and return SD_E_SUCCESS.
If SDDM does not have SDLockDrive interface, just return SD_E_SUCCESS.

[Implementation Note for SDDM]

This interface is optional for application SDDM. If the interface exists, SDEM will call it.

6.9.2 Unlock the Drive

[Format]

```
UINT   SDUnlockDrive(UINT handle);
```

[Argument]

[IN]	UINT	handle	Handle

[Return Value]

0: Success, Positive number: Error code

[Description]

When a specified handle locks a drive, this function unlocks it. If a handle locking the drive is different from the specified handle, this function returns SD_E_UNLOCK_FAILURE.
The handle acquired by the SDInit function is designated as the first argument.

[Remarks]

Before SDFini releases the handle, the drive locked by this function is required to be unlocked by SDUnlockDrive

[Implementation Note for SDEM]

If there is not any SDEM error factor, SDEM behaves like below,
If SDDM has SDUnlockDrive interface, call it and return SD_E_SUCCESS.
If SDDM does not have SDUnlockDrive interface, just return SD_E_SUCCESS.

[Implementation Note for SDDM]

This interface is optional for application SDDM. If the interface exists, SDEM will call it.

7. List of Return Codes

7.1 Return Codes

Table 7-1 : Return Codes List

Return Code		Details
Return Codes	Value	Description
SD_E_SUCCESS	0x0000	Success
SD_E_BAD_VARIABLES	0x1001	Argument error
SD_E_OVER_DRIVELETTER	0x1002	Drive letter A-Z over
SD_E_BUF_NULL	0x1003	I/O buffer null pointer
SD_E_NOT_ENOUGH_MEMORY	0x1004	Not enough memory
SD_E_WP_ERR	0x1005	Write Error (Write Protected)
SD_E_LOCK_FAILURE	0x1006	Drive Lock Failure
SD_E_UNLOCK_FAILURE	0x1007	Drive Unlock Failure
SD_E_DRIVE_LOCKED	0x1008	Drive is locked by another handle.
SD_E_MEDIA_CHANGE	0x1009	Media is removed/inserted.
SD_E_FUNC_NOT_SUPPORTED	0x100A	Function is not supported
SD_E_CARD_INVALID	0x100B	Invalid card. Card does not support the function.
SD_E_SYS_INITIALIZED	0x1081	System already initialized
SD_E_SYS_NOT_INITIALIZED	0x1082	System not initialized
SD_E_HANDLE_OPENED	0x1101	When SDSysFini() is called, the handle remains open.
SD_E_HANDLE_INVALID	0x1102	A specified handle is invalid.
SD_E_HANDLE_FULL	0x1103	Handles cannot be opened any more.
SD_E_ID_INVALID	0x1181	Invalid id number
SD_E_ID_OVERFLOW	0x1182	Id number over
SD_E_DEVICE_ERR	0x12xx*1	Device error
SD_E_ERR	0x13xx*2	The other error

*1 Any number between 0x1200 - 0x12FF (SDDM for vendors)
*2 Any number between 0x1300 - 0x137F (SDDM for vendors)
Any number between 0x1380 - 0x13FF (SDEM)

7.2 Table of Return Codes and Functions

7.2.1 SD Extension Manager (SDEM)

Table 7-2 : Return Codes of SD Extension Manager (SDEM)

SD Extension Manager																							
Return Code (Value)	SDSysInit	SDSysFini	SDInit	SDFini	SDSysDrive	SDEnumSDDrive	SDGetVersion	SDGetCapability	SDGetCSD	SDGetCID	SDGetSDStatus	SDGetSCR	SDGetOCR	SDReadExSingle	SDReadExMulti	SDWriteExSingle	SDWriteExMulti	SDSetCallBack	SDErase	SDGenCmd	SDLockDrive	SDUnlockDrive
0x0000	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
0x1001			O		O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
0x1002			M																			
0x1003							O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
0x1004	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
0x1005																						
0x1006																					M	
0x1007																						M
0x1008									O	O	O	O	O	O	O	O	O	O	O	O	M	M
0x1009																						
0x100A																						
0x100B																						
0x100C																						
0x1081	O																					
0x1082		O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
0x1101		O																				
0x1102				M	M		M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
0x1103			O																			
0x1181																						
0x1182																						
0x12xx																						
0x13xx	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O

M: Mandatory, O: Option

Notice: SDEM does not support the function in the slashed column

7.2.2 SD Device Manager (SDDM)

Table 7-3 : Return Codes of SD Device Manager (SDDM)

SD Device Manager																						
Return Code (Value)	SDSysInit	SDSysFini	SDInit	SDFini	SDSysDrive	SDEnumSDDrive	SDGetVersion	SDGetCapability	SDGetCSD	SDGetCID	SDGetSDStatus	SDGetSCR	SDGetOCR	SDReadExSingle	SDReadExMulti	SDWriteExSingle	SDWriteExMulti	SDSetCallBack	SDErase	SDGenCmd	SDLockDrive	SDUnlockDrive
0x0000	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
0x1001			O				O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
0x1002			M																			
0x1003								O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
0x1004	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
0x1005																						
0x1006																					M	
0x1007																						M
0x1008									O	O	O	O	O	O	O	O	O	O	O	O	M	M
0x1009			O	O																		
0x100A																						
0x100B																						
0x100C																						
0x1081	O																					
0x1082		O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
0x1101		O																				
0x1102				M			M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
0x1103			O																			
0x1181																						
0x1182																						
0x12xx	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
0x13xx	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O

M: Mandatory, O: Option

Notice: SDDM does not support the function in the slashed column

8. File Name and Location

8.1 Windows

8.1.1 SD Extension Manager (SDEM)

There are two types of SDEM. One is called common SDEM. The other is called application SDEM. The file name of both common SDEM and application SDEM shall be “sdem.dll”.

8.1.2 SD Device Manager (SDDM)

The file name of SDDM for each drive is determined in the following manner.

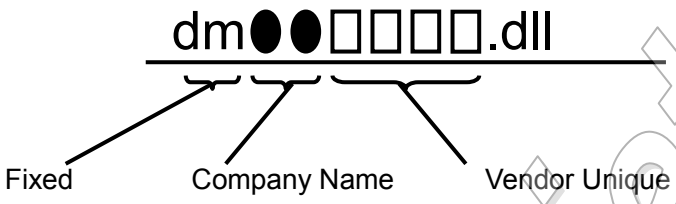

Figure 8-1 : File name manner of DLL for Device Manager

- The first two bytes contain the fixed value of “dm.”
- Company Name contains two-byte company name code, which is provided by the SDA.
- Vendor Unique contains a maximum of four-byte value which is determined by the SDDM provider.
- Its extension is “dll.”

8.1.3 Location

This section is a blank in the simplified specification.

8.1.4 File Versioning

SDEM or SDDM should inform their file version to users. For example, checking file version in DLL properties application may recognize the DLL is old and update the DLL. Detail of operation and deployment is out of this specification’s scope.

9. Usage

Chapter 9 is a blank in the simplified specification.

10. Additional Information

Chapter 10 is a blank in the simplified specification.

Appendix A (Normative) : Reference

A.1 Reference

This specification refers the following documents.

- Part 1 Physical Layer Specification Ver.4.20 or later
- Part A2 SD Host Controller Specification Ver.4.20 or later

Appendix B (Normative) : Special Terms

B.1 Terminology

SD Device Manager	A module to provides an interface for interacting with SDEM and underlying SD Host Controller hardware directly or via a driver to control the SD Host Controller hardware.
SD Extension Manager	A module to provides a common interface for interacting with multiple applications and attached multiple SD Device Managers
SD Extension System	A system to consist of one SD Device Manager and one and more SD Extension Managers

B.2 Abbreviations

API	Application Programming Interface
ATA	Advanced Technology Attachment
CID	Card IDentification
CSD	Card Specific Data
DLL	Dynamic Link Library
OCR	Operation Condition Register
SCR	SD Card Configuration Register
SDDM	SD Device Manager
SDEM	SD Extension Manager
USB	Universal Serial Bus
PnP	Plug and Play

B.3 Expression for function

API's definition uses following data types.

BYTE	Unsigned char
UCHAR	Unsigned char
UINT	Unsigned int
ULONG	Unsigned long