Optical Disc Archive

ODA Drive SDK Guide

Version 4.1.1
December 22th 2017

Sony Corporation

Optical Disc Archive

ODA Drive SDK Guide

Version 4.1.1

December 22th 2017

Conditions of Publication

COPYRIGHT

DISCLAIMER

LICENSING

CLASSIFICATION

Table of Contents

1 General

1.1 Outline of ODA Drive SDK

An ODA drive (e.g. ODS-D55U) can mount an ODA medium (e.g. ODC1500R) which contains 12 optical discs and a cartridge memory (CM) inside. And the dedicated ODAFS driver provides that general file I/O interface as a usual volume (ODA volume). There are some limitations and restrictions to control ODA volumes, and there are also effective ways to read or write files by the general file I/O interface.

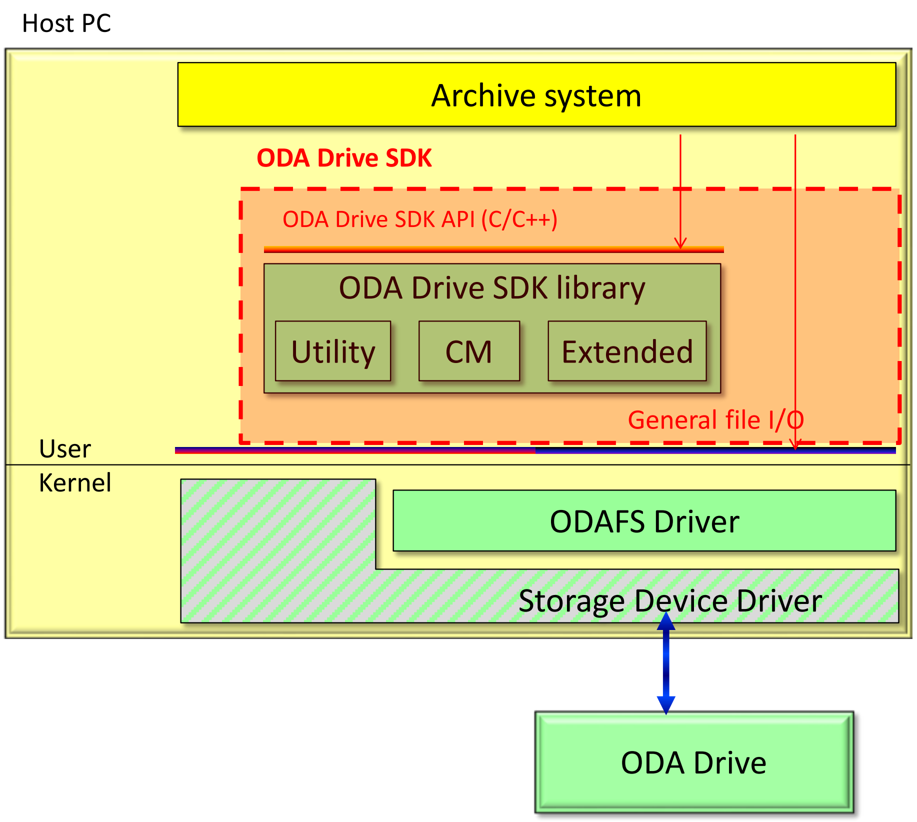
Figure 1-1 Optical Disc Archive Drive Unit & Cartridge

Sony also provides useful and original API, which contains the utility functions such as re-formatting an ODA volume, the accessibility for CM, and the inquiry methods of original or extended information of ODA drives or volumes. This API is called ODA Drive SDK API, and its module is ODA Drive SDK library.

This document also describes how to use ODA Drive SDK API of ODA Drive SDK library.

Finally, ODA Drive SDK includes some sample codes which are command base C++ projects controlling ODA drives or volumes via the general file I/O or ODA Drive SDK API above.

1.2 Scope of ODA Drive SDK



1.3 Definition of terms and acronyms

Cartridge: Physical package of storage unit which contains 12 optical discs inside.

Volume: Logical storage unit which contains files or directories accessible from a root directory.

Media / Medium: Collective term of cartridge and volume.

1.4 Notation

1.4.1 Numerical Notation

Numbers in decimal notation are represented as a sequence of decimal digits with no suffix, while numbers in hexadecimal notation are represented as a sequence of hexadecimal digits suffixed by “h”.

1.4.2 Arithmetic notation

The notation Int(x) shall mean the integer part of x.

The notation AlignDown(a,b) shall mean b×Int(a/b) , where a and b are integers.

1.4.3 Units

In general, for example, when 1KB is expressed, it is ambiguous whether it means power of ten (10^3 bytes) or power of two (2^{10} bytes). To clarify them, this document defines as follows:

1 kB	= 10^3 bytes,	1 KiB	= 2^{10} bytes
1 MB	= 10^6 bytes,	1 MiB	= 2^{20} bytes
1 GB	= 10^9 bytes,	1 GiB	= 2^{30} bytes
1 TB	= 10^{12} bytes,	1 TiB	= 2^{40} bytes

2 General File I/O Interface

2.1 Common for All Operating Systems

2.1.1 Numerical Limitation

2.1.1.1.1 Maximum Number of Files

Maximum number of files recorded on an ODA volume, inclusive of directories, is 60000 (parity on), or 240000 (parity off), where the root directory is also counted as one.

2.1.1.1.2 Maximum depth of Directories

Maximum depth of directories of the file node is 64, where the root directory is also counted as one. For example, the depth of "E:\test.txt" is 2.

2.1.2 Naming Conversion

All characters in a filename, directory name, and volume label (i.e. logical volume identifier) shall be expressed by Unicode 2.0. In addition, the available character code range is U+0 to U+10FFFF except shown the table below.

Code	Character
U+0000 - U+001F	
U+0022	" (double quoatation)
U+002A	* (asterisk)
U+002F	/ (slash)
U+003A	: (colon)
U+003C	< (less than)
U+003E	> (greater than)
U+003F	? (question mark)
U+005C	\(back slash, or Yen mark)
U+007C	| (vertical bar)
U+007F	(DEL)

2.1.2.1 Length of File Names

The maximum length of a filename or a directory name is 127 characters.

The maximum length of a volume label is 63 characters.

Within a filename, directory name, or volume label, a character expressed by surrogate pair is counted as two characters.

2.1.3 File Access Restriction

2.1.3.1 Write Protection

To perform sequential recording, the following restrictions are applied for the write operation:

- Only one file can be write-open simultaneously. When a file is write-open, another write-open operation will be rejected with an error.
- Only new file or the file which size is zero can be write-open. In other words, a non-zero size existing file can be neither over-written nor appended.
- Seek operation is not allowed for write-open file. All the data shall be written sequentially.

2.1.3.2 Read Operation

The following one restriction applies for the read operation:

- Write-open file cannot be read-open. When try to read-open a file and the file is already write-open, the read-open request will be rejected with an error. The file shall be write-closed before read-open

Maximum number of file handles for reading is up to Operation System limitations.

Reading file with seeking (random reading) is also available.

2.1.3.3 File Attributes

File attributes such as ReadOnly/Hidden/System/Archive can be set as usual file system. However, there are some restrictions depend on the running OS.

2.1.4 Recommendation and Tips

2.1.4.1 Detect Write Error

The application which writes a file to ODA volume shall check the returned error of create, write, close function. <u>Furthermore, the application shall check that the written file size in ODA volume is equivalent to the source file size after completing the file writing</u>. If the file is recorded with ODADriveSDK_FILE_PACKED_WRITE flag by ODADriveSDK_SetFileControlOptionEx (), the application shall check the file size after when the application write another file WITHOUT ODADriveSDK_FILE_PACKED_WRITE flag by ODADriveSDK_SetFileControlOptionEx(), or call ODADriveSDK_FlushVolumeBuffers().

Because, ODAFS driver always uses internal cache buffer for writing file, even if user indicated to disable the cached write. Such internal cache buffer will be flushed at the time of closing file handle. And, it is impossible to return such error at the time of closing file handle by ODAFS driver. ODAFS driver will truncate such “error file” size till the size recorded successfully.

It means “error file” size will be shorter than source file size. In other words, if the written file in ODA volume size is equivalent to source file size, ODAFS driver has not detected any device error during the writing process. Please, note that verification of the written file stream is another work. For that purpose, “verify write mode” is provided for ODA volume. It can be set by ODA Drive SDK.

2.1.4.2 Remaining Volume Size

The application which tries to write a file to ODA volume shall check the remaining size of ODA volume is enough before creating the file. ODAFS driver returns the volume remaining size as a file will be able to use that size without medium error occurrence.

Thus, the application shall check the volume remaining size is larger than the file size, and take a margin for device error occurrence as whichever larger: 10% of the writing file size or 128MiB.

And the application shall check the remaining volume size again at next file writing time.

2.1.4.3 Simultaneous File Access

It is restricted to create multiple write file handles simultaneously on the same ODA volume.

On the other hand, it is possible to create or open read file handles even while another write file handle is opened on the same ODA volume. The files on reading or writing may be recorded on different discs in the ODA medium. Therefore, if the application issues read or write commands for each file handle in turn to the same ODA volume, it causes serious performance down because of frequent disc changing in the ODA drive.

The application should make those commands sequence together and order them sequentially to issue them for each file. Ideally, the application should have just only one file handle at the same time, and should create or open file handle after closing another one in one by one manner for an ODA volume.

2.1.4.4 Flushing and Disaster Recovery Policy of ODAFS driver

ODAFS driver caches FS information internally, and flush it to ODA medium at following time:

- Closing the write file handle
- Approx. 5 seconds after the end of other changings:
- Closing file handles which has been used as creating directory, or deleting / renaming / moving / changing attributes of file or directory.
- Changing volume label
- Ejecting the ODA medium
- Finalizing, Re-formatting, Roll-backing the ODA volume

While application is changing something for an ODA volume, and if serious trouble such as power down of the ODA drive is occurred before finishing of above FS flushing, ODAFS driver will roll-back to the last roll-back point which has been recorded by flushing FS information successfully at next mounting time.

It means the writing file will be committed to ODA medium synchronously at the time of the handle closing. However, the other changing of ODA volume (e.g. create directory, delete file.. and so on) will not be committed to ODA medium until next FS flushing even after closing of those handles.

2.2 Local Restrictions and Specifications of general API

2.2.1 Windows

The caller process shall run as administrator. Otherwise, some of functions will not work.

2.2.2 Macintosh OSX

When a cartridge is mounted, cartridge is mounted automatically under /Volume. The directory name under /Volumes is used to be “Volume Name”.

T.B.D

2.2.3 Linux

When a cartridge is inserted, the behavior of mount depends on the configuration of a system and the model of ODA drive.

1. A system which Udev is installed
   - ODS-D55U、ODS-D77U
     The cartridge is mounted automatically under /media. The directory name under /media is used to be “Volume Name”.

   - ODS-D77F
     The cartridge is automatically recognized, although not mounted automatically. In order to mount, it is necessary to click the icon named "Volume Name" on Nautilus or run the following command on a terminal.
     ```
     >udisks --mount /dev/sdx
     ```

2. A system which Udev is not installed
   In order to mount, it is necessary to run the following command on a terminal.
   e.g.
   ```
   >mount -t odaudf /dev/sdx /mnt/mnt_point
   ```

3 ODA Drive SDK

ODA Driver SDK is a utility library for software vendors who support ODA drives. This SDK incudes the features to get drive and volume information and file location on the disc. Also, it supports the reading and writing of Cartridge Memory, media formatting, enable or disable software write protect and finalization to write once media.

3.1 Software Requirements

3.1.1 Windows Environment

ODA Driver SDK supports the following Operating Systems

Windows 7 SP1 32/64bits
Windows 8 32/64 bits
Windows 8.1 32/64bits
Windows 10 32/64bits
Windows Server 2008 R2
Windows Server 2012
Windows Server 2012 R2

Sony provides 32bits and 64bits dll (dynamic link library) include files and import libraries for release and debug build. Also, Sony provides sample code how to use SDK. The project file and solution file of sample codes are for Visual Studio 2010.

3.1.2 Macintosh OSX Environment

Macintosh version will be supported in the feature version. We have a plan to support the following Operating Systems.

OSX 10.6.8 32bits (Snow Leopard)
OSX 10.7.5 32/64bits (Lion)
OSX 10.8.4 64bits (Mountain Lion)
OSX 10.9.5 64bits (Mavericks)
OSX 10.10.5 64bits (Yosemite)
OSX 10.11.2 64bits (El Capitan)

Sony will provide 32bits and 64bits universal binary framework. Also, Sony will provide sample codes. The project files of sample codes are for Xcode 4.5 or later.

3.1.3 Linux Environment

Linux version supports the following distributions.

Red Had Enterprise Linux 6.2/6.3/6.4/6.5/6.6/7.0/7.1 64bits for Intel Platform
Sony will provide 64bits so files, include files. Sony will also provide sample codes and makefile.

An application which links ODA Driver SDK for Linux shall run with root privilege, since the SDK issues SCSI commands to SCSI device directly.

3.2 Contents of SDK

3.2.1 Windows Environment

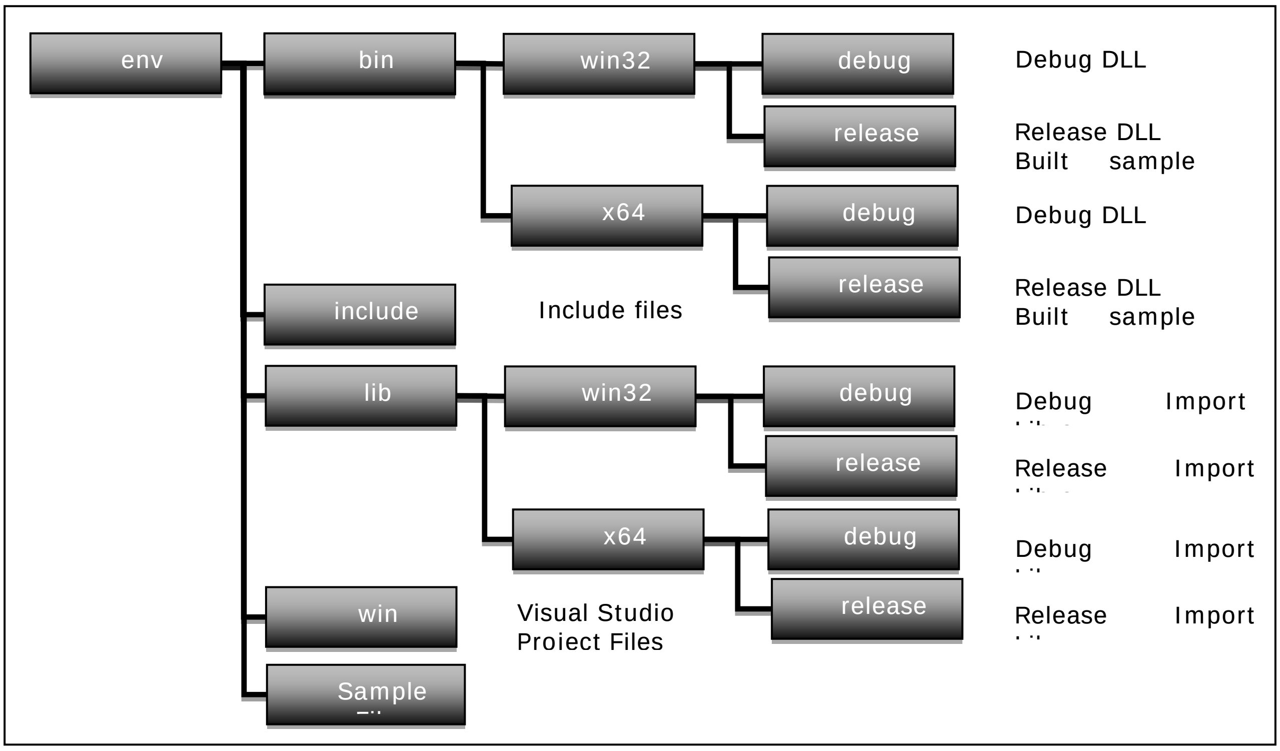


Figure 3-1 Windows Platform Contents of SDK

3.2.2 Macintosh OSX Environment

T.B.D.

3.2.3 Linux Environment

There are three components in zip archive.

libodadrivesdk-x.x.x-x.el6.x86_64.rpm : SDK package
libodadrivesdk-devel-x.x.x-x.el6.x86_64.rpm : Development package
example.tar.gz : Sample codes

Following are instructions for installing SDK package and Development package.

```
[Confirm the current rpms]
> rpm -qa | grep libodadrivesdk
> libodadrivesdk-x.x.x-x.el6.x86_64
> libodadrivesdk-devel-x.x.x-x.el6.x86_64

[Uninstall the old SDK rpms with the following orde]r
>rpm -e  libodadrivesdk-devel-x.x.x-x.el6.x86_64
>rpm -e libodadrivesdk-x.x.x-x.el6.x86_64

[Install the new SDK rpms with the following order]
> rpm -ivh libodadrivesdk-x.x.x-x.el6.x86_64.rpm
> rpm -ivh libodadrivesdk-devel-x.x.x-x.el6.x86_64.rpm
```

Following are instructions for compiling sample codes.

```
[Compiling sample codes]
>cd DriveSDKx.xx/ODADriveSDK_Vxxxx_Linux/env/example
>./configure
>make.
```

3.3 API Summary

3.3.1 General operations

3.3.1.1 SDK Version

An application can get the ODA Drive SDK version by *ODADriveSDK_GetVersion()*. Major/Minor/Update version value is equivalent to Optical Disc Archive Software (ODA driver/utility) corresponding to this SDK. Internal version value is given for this ODA Drive SDK originally.

```
#define ODA_DRIVE_SDK_VERSION_MAJOR          4
#define ODA_DRIVE_SDK_VERSION_MINOR          1
#define ODA_DRIVE_SDK_VERSION_UPDATE         0
#define ODA_DRIVE_SDK_VERSION_INTERNAL       4
#define ODA_DRIVE_SDK_VERSION_STRING         ODADriveSDK_INITSTRING("3.2.0.2")
```

Figure 3-2 Version Definitions (example)

```
/**
 * function: ODADriveSDK_GetVersion
 *
 * summary:
 *     Get ODSDriverSDK Version.
 *
 * return: ODSDriverSDK Version.
 */
ODA_DRIVE_SDK_CAPI
ODADriveSDK_CHAR* ODADriveSDK_GetVersion(void);
```

Figure 3-3 GetVersion Function

3.3.1.2 Error Message

An application can get the reason of error by *ODADriveSDK_GetErrorMessage()*. The errCode will be given as return value of this SDK functions.

```
/**
 * function: ODADriveSDK_GetErrorMessage
 *
 * summary:
 *          Get error message of ODADriveSDK.
 *
 * param [in]    errCode: error code which returned from fucntion
 * param [out] errMsgBuff: pointer to buffer of error msg
 * param [in]    errMsgBuffLength: buffer length of error msg
 *
 * return:
 */
ODA_DRIVE_SDK_CAPI void
ODADriveSDK_GetErrorMessage(
                         uint64_t errCode,
                         char *errMsgBuff,
                         uint32_t errMsgBuffLength
);
```

Figure 3-4 GetErrorMessage Function

3.3.1.3 Operational Mode

An application can get operational mode by *ODADriveSDK_GetOperationalMode()*, set them by *ODADriveSDK_SetOperationalMode()*, and reset them as factory settings by *ODADriveSDK_ResetOperationalMode()*.

The parameter of operational mode is registered in the running system (PC).

It means when those parameters are set, the changes effect to all the drives/volumes which connected to the running system.

To effect those parameters, application shall eject the cartridge once by *ODADriveSDK_DoEject()* for example, and re-inject the cartridge again.

```
/**   operational mode */
enum ODADriveSDK_OPERATIONAL_MODE_NAME
{
        ODADriveSDK_DEFAULT_VOLUME_TYPE = 0,
        ///< default volume type. // ODADriveSDK_VOLUME_TYPE value; # refer
ODADriveSDK_VOLUME_TYPE declaration.
        ODADriveSDK_WRITE_VERIFY = 1,
        ///< write-verify. // uint32_t value; # No verify == 0, Verify == 1. Reserved == others.
        ODADriveSDK_DRIVES_REC_INHIBIT = 2,
        ///< make the drives rec-inhibit. // uint32_t value; # No restrictions == 0, Rec Inhibit == 1. Reserved
== others.
        ODADriveSDK_DEFAULT_FS_SYNC = 3
        ///< synchronize management data to media immediately after completion of writing files.   //
ODADriveSDK_FILE_CONTROL_OPTION_OF_FS_FLUSH value; # refer
ODADriveSDK_FILE_CONTROL_OPTION_OF_FS_FLUSH declaration.
};
```

Figure 3-5 OperationalModeName Enum

```
/**
 * function: ODADriveSDK_GetOperationalMode
 *
 * summary:
 *       Get current operational mode.
 *
 * param [in]   modeName: name of operational mode.
 * param [out] value: pointer to value which size depends on modeName (refer
ODADriveSDK_OPERATIONAL_MODE_NAME).
 * param [in]   valueBufferSize: buffer size of value.
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_GetOperationalMode(
                enum ODADriveSDK_OPERATIONAL_MODE_NAME modeName,
                void* value, uint64_t valueBufferSize
);
```

Figure 3-6 GetOperationalMode Function

```
/**
  * function: ODADriveSDK_SetOperationalMode
  *
  * summary:
  *           Set current operational mode.
  *
  *           param [in]    modeName: name of operational mode.
  *    param [in]    value: pointer to value which size depends on modeName (refer
ODADriveSDK_OPERATIONAL_MODE_NAME).
  *           param [in]    valueBufferSize: buffer size of value.
  *
  *    return: zero if successful, or error code otherwise.
  */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_SetOperationalMode(
                              enum ODADriveSDK_OPERATIONAL_MODE_NAME modeName,
                              const void* value, uint64_t valueBufferSize
);
```

Figure 3-7 SetOperationalMode Function

```
/**
  * function: ODADriveSDK_ResetOperationalMode
  *
  * summary:
  *             Reset current operational mode as factory settings.
  *
  * return: zero if successful, or error code otherwise.
  */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_ResetOperationalMode(void);
```

Figure 3-8 ResetOperationalMode Function

3.3.1.4 Export Drive/Driver logs

An application can get drive/driver logs by ODADriveSDK_GetDrivelog()/GetDriverLog().

To get a drive log by ODADriveSDK_GetDrivelog(), eject a cartridge from drive before it, otherwise it failed. Check the presence or absence of a cartridge by ODADriveSDK_CheckMediaExist() before calling ODADriveSDK_GetDrivelog(). If a cartridge exists, eject it by ODADriveSDK_DoEject(). The drive will be restarted after the completion of ODADriveSDK_GetDrivelog(). ODADriveSDK_GetDrivelog() works only for ODS-D77U/ODS-D280U.

Using ODADriveSDK_GetDriverlog(), collect driver log files in where these are saved to the designated directory. If ODADriveSDK_DRIVER_LOG_WITH_SYSTEM flag is set, system information will be included.

```
/**
 * function: ODADriveSDK_GetDriveLog
 *
 * summary:
 *     Get a drive log.
 *
 * param[in] drive:
 *                    Drive path of target
 *                    In case of Windows, set "G:", or "G:\" (case insensitive).
 *                    In case of Mac, set "deviceId".
 *                    In case of Linux, set "/dev/sdX", or"/dev/sgX".
 * param[in] outputFilePath:
 *                    Output file path to save the drive log
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI unit64_t
ODADriveSDK_GetDriveLog(
            const ODADriveSDK_CHAR* drive,
            const ODADriveSDK_CHAR* outputFilePath
);
```

Figure 3-9 GetDriveLog Function

```
enum ODADriveSDK_DRIVER_LOG_FLAG
{
        ODADriveSDK_DRIVER_LOG_ONLY = 0,                ///< Get driver logs only.
        ODADriveSDK_DRIVER_LOG_WITH_SYSTEM = 1       ///< Get driver logs with system information.
};
```

Figure 3-10 DriverLogFlag Enum

```
/**
  * function: ODADriveSDK_GetDriverLog
  *
  * summary:
  *     Get driver logs.
  *
  * param[in] driverLogFlag:
  *                     Option flag
  * param[in] outputDirPath:
  *                     Output directory path to collect and save driver logs
  *
  * return: zero if successful, or error code otherwise.
  */
ODA_DRIVE_SDK_CAPI unit64_t
ODADriveSDK_GetDriverLog(
            enum ODADriveSDK_DRIVER_LOG_FLAG driverLogFlag,
            const ODADriveSDK_CHAR* outputDirPath
);
```

Figure 3-11 GetDriverLog

3.3.2 Drive / Media operations

3.3.2.1 Medium Information

An application can get the basic information about a medium in the drive by *ODADriveSDK_GetInformation()*.

The MountStatus ODADriveSDK_MOUNT_STATUS_NOT_READ is available for Linux only.

```
#define ODADriveSDK_MOUNT_STATUS_NO_ERROR            0x00000000
#define ODADriveSDK_MOUNT_STATUS_DRIVER_ERROR        0x80000001
#define ODADriveSDK_MOUNT_STATUS_DEVICE_ERROR        0x80000002
#define ODADriveSDK_MOUNT_STATUS_SYSTEM_GUARD        0x80000003
#define ODADriveSDK_MOUNT_STATUS_NOT_READY            0x80000004
#define ODADriveSDK_MOUNT_STATUS_BLANK_MEDIA         0x80000010
#define ODADriveSDK_MOUNT_STATUS_UNSUPPORTED_MEDIA         0x80000020
#define ODADriveSDK_MOUNT_STATUS_CORRUPTED_MEDIA           0x80000030
#define ODADriveSDK_MOUNT_STATUS_UNSUPPORTED_VOLUME        0x80000100
#define ODADriveSDK_MOUNT_STATUS_UNKNOWN_VOLUME            0x80000200
#define ODADriveSDK_MOUNT_STATUS_INCONSISTED_VOLUME_UNRECOVERABLE_VERSION
            0x80000300
#define ODADriveSDK_MOUNT_STATUS_INCONSISTED_VOLUME_UNRECOVERABLE
            0x80000400
```

Figure 3-12 MountStatus Definitions

```
#define ODADriveSDK_WRITE_PROTECT_DEVICE_CONDITIONS        0x00000001
#define ODADriveSDK_WRITE_PROTECT_MEDIA_CONDITIONS         0x00000100
#define ODADriveSDK_WRITE_PROTECT_UNRECORDABLE_MEDIA       0x00000200
#define ODADriveSDK_WRITE_PROTECT_MEDIA_SETTINGS           0x00000400
#define ODADriveSDK_WRITE_PROTECT_WORN_OUT_MEDIA           0x00000800
#define ODADriveSDK_WRITE_PROTECT_FINALIZED_MEDIA          0x00001000
#define ODADriveSDK_WRITE_PROTECT_CM_SETTINGS              0x00002000
#define ODADriveSDK_WRITE_PROTECT_CM_CONDITIONS            0x00004000
#define ODADriveSDK_WRITE_PROTECT_ROLLBACKED_VOLUME        0x00020000
#define ODADriveSDK_WRITE_PROTECT_USER_SETTINGS            0x00040000
#define ODADriveSDK_WRITE_PROTECT_DRIVER_RESTRICTION       0x00080000
#define ODADriveSDK_WRITE_PROTECT_TEMPORAL_LOCK            0x00100000
```

Figure 3-13 WriteProtectReasonFlags Definitions

```
#define ODADriveSDK_ACCESS_MODE_NORMAL                     0
#define ODADriveSDK_ACCESS_MODE_RAW                        2
```

Figure 3-14 AccessMode Definitions

```
#define ODADriveSDK_MEDIUM_TYPE_UNKNOWN     (0x00) // Unknown medium
#define ODADriveSDK_MEDIUM_TYPE_BD_SL_R     (0xE1) //BD Single Layer Recordable
#define ODADriveSDK_MEDIUM_TYPE_BD_SL_RE    (0xE2) //BD Single Layer Rewritable
#define ODADriveSDK_MEDIUM_TYPE_BD_DL_R     (0xE3) //BD Dual Layer Recordable
#define ODADriveSDK_MEDIUM_TYPE_BD_DL_RE    (0xE4) //BD Dual Layer Rewritable
#define ODADriveSDK_MEDIUM_TYPE_BD_TL_RE    (0xE6) //BD Triple Layer Rewritable
#define ODADriveSDK_MEDIUM_TYPE_BD_QL_R     (0xE7) //BD Quad Layer Recordable
#define ODADriveSDK_MEDIUM_TYPE_AD_TL_R     (0xF1) //AD Triple Layer Recordable
```

Figure 3-15 MediumType Definitions

```
enum ODADriveSDK_VOLUME_TYPE
{
        ODADriveSDK_VOLUME_TYPE_PARITY_ON = 0,
///< volume type=0: Maximum number of files or directories is 60000. Parity stream will be generated
background while file recording.
        ODADriveSDK_VOLUME_TYPE_PARITY_OFF = 1
///< volume type=1: Maximum number of files or directories is 240000. No parity stream.
};
```

Figure 3-16 VolumeType Enum

```
#define ODADriveSDK_DISC_USAGE_CONDITION_WRITABLE                0
#define ODADriveSDK_DISC_USAGE_CONDITION_RECOMMEND_NEW_DISC      1
#define ODADriveSDK_DISC_USAGE_CONDITION_WRITE_PROTECTED         2
```

Figure 3-17 DiscUsageCondition Definitions

```
struct ODA_DRIVE_SDK_CAPI ODADriveSDK_DiscUsageInfo
{
        uint8_t   index;        ///< Index of the disc written in progress
        uint8_t   condition; ///< Condition of the disc written in progress
        uint64_t capacity;   ///< Capacity of the disc written in progress[bytes]
        uint64_t available; ///< Available space size of the disc written in progress [bytes]
        uint64_t used;          ///< Used space size off the disc written in progress (= capacity - available)
[bytes]
};
```

Figure 3-18 DiscUsageInfo Structure

```
struct ODA_DRIVE_SDK_CAPI ODADriveSDK_BasicInfo
{
        uint8_t          serialNumber[ODA_SERIAL_NUMBER_LENGTH]; ///< serial number of the
cartridge
        uint32_t         initializationCount;      ///< initialization count of the volume
        uint32_t         initializationId;         ///< id set at the time of volume creation
        uint32_t         modificationId;           ///< id set at the time of volume modification
        uint8_t          numberOfDiscs;            ///< number of discs in the cartridge
        uint32_t         mountStatus;              ///< mount status of the volume
        uint32_t         writeProtectReason;       ///< flags of write protect reason of the volume and cartridge
        uint32_t         accessMode;               ///< access mode of the volume
        uint8_t          mediumType;               ///< medium typen of the discs and cartridge
         char             vendor[8];                ///< cartridge vendor id
         char             productName[16];          ///< cartridge product name
         char             mediaTypeForLongForm[48]; ///< cartridge type (long form)
         char             mediaTypeForShortForm[16]; /// cartridge type (short form)   enum
ODADriveSDK_VOLUME_TYPE volumeType;  ///< type of the volume
        struct ODADriveSDK_DiscUsageInfo usageInfo;           ///< usage info of the disc written in
progress
};
```

Figure 3-19 BasicInfo Structure

```
/**
 * function: ODADriveSDK_GetInformation
 *
 * summary:
 *     Get basic information of the medium in the drive.
 *
 * param[in] drive:
 *                  Drive path of target
 *                  In case of Windows, set "G:", or "G:\" (case insensitive).
 *                  In case of Mac, set "/Volumes/XXX", "/dev/diskX", or "deviceId".
 *                  I  In case of Linux, set mount point, "/dev/sdX", or"/dev/sgX".
 * param[out] info:
 *                  Basic information of drive
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_GetInformation(
                    const ODADriveSDK_CHAR* drive,
                    struct ODADriveSDK_BasicInfo* info
);
```

Figure 3-20 GetInfomation Function

3.3.2.2 Drive Identifier(Macintosh OSX only)

An application can get identifier of drive connected to Macintosh by ODADriveSDK_GetDriveIdVector. The identifier of drive can be used as “drive” parameter in following APIs.

ODADriveSDK_GetInformation

ODADriveSDK_DoEject

ODADriveSDK_GetDeviceInformation

ODADriveSDK_CheckMediaExist

ODADriveSDK_SetSoftwareWriteProtect

ODADriveSDK_GetSoftwareWriteProtect

```
//* drive identifier */
struct ODA_DRIVE_SDK_CAPI ODADriveSDK_DriveId
{
            char deviceId[17];  ///< drive identifier(null terminal char string)
};

/** drive identifier vector */
struct ODA_DRIVE_SDK_CAPI ODADriveSDK_DriveIdVector
{
            uint32_t driveIdNum;                                                ///< number of drive
identifier
            struct ODADriveSDK_DriveId driveIds[1];                  ///< drive identifier
};
```

Figure 3-21 DriveIdVector Structure

```
/**
 * function: ODADriveSDK_GetDriveIdVector
 *
 * summary:
 *     Get identifier of drive cnnected to the system.
 *
 * param[out] driveIdBuffer:
 *                   Drive identifier
 * param[in]   driveIdBufferSize:
 *                   Buffer size of driveIdBuffer
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
 ODADriveSDK_GetDriveIdVector(
                        struct ODADriveSDK_DriveIdVector* driveIdBuffer,
                        uint64_t driveIdBufferSize
);
```

Figure 3-22 GetDeviceInfomation Function

3.3.2.3 Drive Information

An application can get the drive information such as drive’s model name, serial name and firmware version by ODADriveSDK_GetDeviceInformation and ODADriveSDK_GetDeviceInformationEx. The application can also get the alarm code and hours meters of drive.

Note that ODADriveSDK_GetDeviceInformation dosen't support part of ODADriveSDK_HoursMeter(discGuideCounter, carryMotorHCounter, carryMotorVCounter).

```
struct ODADriveSDK_HoursMeter
{
        uint8_t     dataType; ///< data Structure format type
        union
        {
                //* ODS-D55U (dataType==0) */
                struct
                {
                        uint16_t    dataStructureLength;            ///< data Structure length
                        uint8_t     dataStructureFormatType;        ///< data Structure format type
                        uint32_t    operationTime;                  ///< operation time
                        uint32_t    spindleTime;                    ///< spindle time
                        uint32_t    laser;                          ///< laser
                        uint32_t    selectCount;                    ///< select count
                        uint32_t    seekCount;                      ///< seek count
                        uint32_t    carryCount;                     ///< carry count
                        uint32_t    injectCount;                    ///< inject count
                } Type0;

                //* ODS-D77U/F (dataType==1) */
                struct
                {
                        uint16_t    dataStructureLength;            ///< data Structure length
                        uint8_t     dataStructureFormatType;        ///< data Structure format type
                        uint32_t    operationTime;                  ///< operation time
                        uint32_t    spindleTime;                    ///< spindle time
                        uint32_t    laser0;                         ///< laser parameter 0
                        uint32_t    laser1;                         ///< laser parameter 1
                        uint32_t    selectCount;                    ///< select count
                        uint32_t    seekCount0;                     ///< seek count 0
                        uint32_t    seekCount1;                     ///< seek count 1
                        uint32_t    carryCount;                     ///< carry count
                        uint32_t    injectCount;                    ///< inject count
                        uint32_t    discGuideCounter;               ///< disc guide count
                        uint32_t    carryMotoreHCounter;            ///< carriy motoer H count
                } Type1;

                //* ODS-D280U/F (dataType==2) */
                struct
                {
                        uint16_t    dataStructureLength;            ///< data Structure length
                        uint8_t     dataStructureFormatType;        ///< data Structure format type
                        uint32_t    operationTime;                  ///< operation time
                        uint32_t    spindleTime;                    ///< spindle time
                        uint32_t    laser0;                          ///< laser parameter 0
                        uint32_t    laser1;                         ///< laser parameter 1
                        uint32_t    laser2;                         ///< laser parameter 2
                        uint32_t    laser3;                         ///< laser parameter 3
                        uint32_t    selectCount;                    ///< select count
                        uint32_t    seekCount0;                     ///< seek count 0
                        uint32_t    seekCount1;                     ///< seek count 1
                        uint32_t    seekCount2;                     ///< seek count 2
                        uint32_t    seekCount3;                     ///< seek count 3
                        uint32_t    carryCount;                     ///< carry count
                        uint32_t    injectCount;                    ///< inject count
                        uint32_t    discGuideCounter;           ///< disc guide count
                        uint32_t    carryMotorHCounter;         ///< carrt motor H count
                        uint32_t    carryMotorVCounter;          ///< carrt motor V count

                } Type2;
        } Data;
};
```

Figure 3-23 HoursMeter Structure

```
struct ODADriveSDK_AlarmCode
{
      uint8_t       mainCode;                                   ///< main code
      uint16_t       subCode;                                   ///< sub code (bits 0-11 used)
};
```

Figure 3-24 AlarmCode Structure

```
struct ODA_DRIVE_SDK_CAPI ODADriveSDK_DeviceInfo
{
            char modelName[17];                 ///< drive model name(null terminal char string)
            char serialNumber[17];              ///< drive serial number(null terminal char string)
            char firmwareVersion[5];            ///< drive firmware version(binary)
            struct ODADriveSDK_HoursMeter       hoursMeter;        ///< drive hours meter
            struct ODADriveSDK_AlarmCode        alarmCode;          ///< drive alarm code
};
```

Figure 3-25 DeviceInfo Structure

```
/**
 * function: ODADriveSDK_GetDeviceInformation
 *
 * summary:
 *     Get the drive information whether a cartridge is in or not.
 *
 * param[in] drive:
 *                  Drive path of target
 *                  In case of Windows, set "G:", or "G:\" (case insensitive).
 *                  In case of Mac, set "/Volumes/XXX", "/dev/diskX", or "deviceId".
 *                  In case of Linux, set mount point, "/dev/sdX", or"/dev/sgX".
 * param[out] info:
 *                  Drive information
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_GetDeviceInformation(
                        const ODADriveSDK_CHAR *drive,
                        struct ODADriveSDK_DeviceInfo* info
);
```

Figure 3-26 GetDeviceInfomation Function

```
/**
 * function: ODADriveSDK_GetDeviceInformationEx
 *
 * summary:
 *     Get the drive information whether a cartridge is in or not.
 *
 * param[in] drive:
 *                   Drive path of target
 *                   In case of Windows, set "G:", or "G:\" (case insensitive).
 *                   In case of Mac, set "/Volumes/XXX", "/dev/diskX", or "deviceId".
 *                   In case of Linux, set mount point, "/dev/sdX", or"/dev/sgX".
 * param[out] info:
 *                   Drive information
 * param[in] infoSize
 *                   Buffer size of info
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_GetDeviceInformationEx(
                     const ODADriveSDK_CHAR *drive,
                     struct ODADriveSDK_DeviceInfo* info,
                     uint32_t infoSize
);
```

Figure 3-27 GetDeviceInfomationEx Function

3.3.2.4 Number of Files and Directories in the Volume

An application can get the number of files and directories in the volume by ODADriveSDK_GetFileCount().

```
/**
 * function: ODADriveSDK_GetFileCount
 *
 * summary:
 *     Get total number of files and directories in the volume.
 *     When access mode(in BasicInfo, get with GetInformation()) of the volume is RAW,
 *     NumberOfFiles and NumberOfDirectories are set 0.
 *
 * param[in] drive:
 *                     Drive path of target
 *                     In case of Windows, set "G:", or "G:\" (case insensitive).
 *                     In case of Mac, set "/Volumes/XXX" or "/dev/diskX".
 *                     In case of Linux, set mount point.
 * param[out] numberOfFiles:
 *                     Number of recorded files
 * param[out] numberOfDirectories:
 *                     Number of recorded directorieds
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_GetFileCount(
                         const ODADriveSDK_CHAR *drive,
                         uint32_t *numberOfFiles,
                         uint32_t *numberOfDirectories
);
```

Figure 3-28 GetFileCount Function

3.3.2.5 Software Write Protect

ODA cartridges have a software write protect setting. This software write protect setting is ORing to other write protect settings such as cartridge's write protect switch.

The volume will be toggled read-only/writable, when an application calls ODADriveSDK_SetSoftwareWriteProtect() to set this software write protect setting on/off.

This feature will be kept even after when the cartridge is ejected.

An application can also obtain the current protect setting by ODADriveSDK_GetSoftwareWriteProtect().

```
enum ODADriveSDK_SOFTWARE_WRITE_PROTECT
{
        ODADriveSDK_SOFTWARE_WRITE_PROTECT_OFF = 0,        ///< software write protect off
        ODADriveSDK_SOFTWARE_WRITE_PROTECT_ON = 1          ///< software write protect on
};
```

Figure 3-29 SoftwareWriteProtect Enum

```
/**
 * function: ODADriveSDK_SetSoftwareWriteProtect
 *
 * summary:
 *     Set the software write protect of the cartridge on or off.
 *     The software write protect flag is recorded in CM(cartridge memory).
 *     The cartridge will be ejected automatically by this calling.
 *
 * param[in] drive:
 *                  Drive path of target
 *                  In case of Windows, set "G:", or "G:\" (case insensitive).
 *                  In case of Mac, set "/Volumes/XXX", "/dev/diskX", or "deviceId".
 *                  In case of Linux, set mount point, "/dev/sdX", or"/dev/sgX".
 * param[in] writeProtect:
 *                  If ODADriveSDK_SOFTWARE_WRITE_PROTECT_OFF, write protect is off.
 *                  If ODADriveSDK_SOFTWARE_WRITE_PROTECT_ON, write protect is on.
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t ODADriveSDK_SetSoftwareWriteProtect(
                    const ODADriveSDK_CHAR *drive,
                    enum ODADriveSDK_SOFTWARE_WRITE_PROTECT writeProtect
);
```

Figure 3-30 SetSoftwareWriteProtect Function

```
/**
 * function: ODADriveSDK_GetSoftwareWriteProtect.
 *
 * summary:
 *     Get whether the software write protect of the cartridge is on or off.
 *
 * param[in] drive:
 *                      Drive path of target
 *                      In case of Windows, set "G:", or "G:\" (case insensitive).
 *                      In case of Mac, set "/Volumes/XXX", "/dev/diskX", or "deviceId".
 *                      In case of Linux, set mount point, "/dev/sdX", or"/dev/sgX".
 * param[out] writeProtect:
 *                      If ODADriveSDK_SOFTWARE_WRITE_PROTECT_OFF, write protect is off.
 *                      If ODADriveSDK_SOFTWARE_WRITE_PROTECT_ON, write portect is on.
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t ODADriveSDK_GetSoftwareWriteProtect(
                            const ODADriveSDK_CHAR *drive,
                            enum ODADriveSDK_SOFTWARE_WRITE_PROTECT *writeProtect
);
```

Figure 3-31 GetSoftwareWriteProtect Function

3.3.2.6 Cartridge type in the Drive

An application can get the type of cartridge in the drive by ODADriveSDK_GetCartridgeType().

```
enum ODADriveSDK_CARTRIDGE_TYPE
{
            ODADriveSDK_CRTRIDGE_TYPE0 = 0, ///< Generation1 type cartridge
            ODADriveSDK_CRTRIDGE_TYPE1 = 1 ///< Generation2 type cartridge
           ODADriveSDK_CARTRIDGE_UNSUPPORTED = -1
};
```

Figure 3-32 CartridgeType Enum

```
/**
 * function: ODADriveSDK_GetCartridgeType
 *
 * summary:
 *     Get the type of cartridge in the drive.
 *
 * param[in] drive:
 *                  Drive path of target
 *                  In case of Windows, set "G:", or "G:\" (case insensitive).
 *                  In case of Mac, set "/Volumes/XXX", "/dev/diskX", or "deviceId".
 *                  In case of Linux, set mount point, "/dev/sdX", or"/dev/sgX".
 * param[out] cartridgeType:
 *                  If ODADriveSDK_CRTRIDGE_TYPE0, Geneartion1 type catridge exis in the drive.
 *                  If ODADriveSDK_CRTRIDGE_TYPE1, Generation2 type catridge exis in the drive.
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t ODADriveSDK_GetCartridgeType(
                  const ODADriveSDK_CHAR *drive,
                  enum ODADriveSDK_CARTRIDGE_TYPE *cartridgeType
);
```

Figure 3-33 GetCartridgeType Function

3.3.2.7 Media existence in the Drive

An application can check a cartridge existence in the drive by ODADriveSDK_CheckMediaExist().

```
enum ODADriveSDK_IS_CARTRIDGE_EXISTED
{
        ODADriveSDK_NO_CARTRIDGE = 0,               ///< there is no cartridge in the drive
        ODADriveSDK_CARTRIDGE_EXIST = 1             ///< a cartridge exists in the drive
};
```

Figure 3-34 IsCartridgeExisted Enum

```
/**
 * function: ODADriveSDK_CheckMediaExist
 *
 * summary:
 *     Check whether a catridge exists or not in a drive.
 *     When the cartridge is on the way to injecting or ejecting,
 *     this function returns as existing.
 *
 * param[in] drive:
 *                 Drive path of target
 *                 In case of Windows, set "G:", or "G:\" (case insensitive).
 *                 In case of Mac, set "/Volumes/XXX", "/dev/diskX", or "deviceId".
 *                 In case of Linux, set mount point, "/dev/sdX", or"/dev/sgX".
 * param[out] isExist:
 *                 If ODADriveSDK_CARTRIDGE_EXIST, a catridge is in the drive.
 *                 If ODADriveSDK_NO_CARTRIDGE, no cartridge is in the drive.
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t ODADriveSDK_CheckMediaExist(
                    const ODADriveSDK_CHAR *drive,
                    enum ODADriveSDK_IS_CARTRIDGE_EXISTED *isExist
);
```

Figure 3-35 CheckMediaExist Function

3.3.2.8 Eject Cartridge from Drive

An application can eject the cartridge from the specified drive by ODADriveSDK_DoEject().

This function will wait until the cartridge has been ejected completely. It means the cartridge can be removed from the cartridge slot of the drive physically after this function calling.

```
/**
  * function: ODADriveSDK_DoEject
  *
  * summary:
  *     Eject a cartridge from the drive safety.
  *
  * param[in] drive:
  *                     Drive path of target
  *                     In case of Windows, set "G:", or "G:\" (case insensitive).
  *                     In case of Mac, set "/Volumes/XXX", "/dev/diskX", or "deviceId".
  *                     In case of Linux, set mount point, "/dev/sdX", or"/dev/sgX".
  *
  * return: zero if successful, or error code otherwise.
  */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_DoEject(
                        const ODADriveSDK_CHAR *drive
);
```

Figure 3-36 DoEject Function

3.3.2.9 Re-formatting the Cartridge

An application can delete all files and directories and re-initialize volume by ODADriveSDK_DoAllDelete ().

Caller shall set two parameters: formatMethod and volumeType.

If ODADriveSDK_BACKTRACK is set to formatMethod parameter, the remaining size of the volume will be regained to initial size. ODADriveSDK_BACKTRACK can be set only for rewritable medium, and cannot be set for write once (recordable) medium. When ODADriveSDK_BACKTRACK is set, volumeType parameter is effect.

If ODADriveSDK_NO_BACKTRACK is set to formatMethod parameter, the remaining size will not be gained. The volumeType parameter will be ignored, and the new re-formatted volume will inherit volumeType from previous volume.

This API may take a time. When you set callback function, API calls callback function with the progress periodically.

After this API is completed, the volume is mounted on Windows and Macintosh OS X. On the other hand, the volume is unmounted on Linux.

```
enum ODADriveSDK_PROCESSING_REQUEST
{
        ODADriveSDK_REQUEST_CANCEL = 0,                     ///< cancel processing
        ODADriveSDK_REQUEST_CONTINUE = 1            ///< continue processing
};
```

Figure 3-37 ProcessingRequest Enum

```
/**
 * function: ODADriveSDK_AllDeleteCallback
 *
 * summary:
 *          User defined callback function called from ODADriveSDK_DoAllDelete.
 *          User can check formatting progress by deleting disc number.
 *          User can select whether continue or cancel for processing by return value.
 *
 * param [in] discNum:
 *                  Number of deleted discs
 * param [in] param:
 *                  User defined parameter set at calling ODADriveSDK_DoAllDelete
 *
 * return: ODADriveSDK_REQUEST_CONTINUE to continue, or ODADriveSDK_REQUEST_CANCEL to
cancel.
 */
typedef enum ODADriveSDK_PROCESSING_REQUEST
(*ODADriveSDK_AllDeleteCallback)(
                    uint32_t  discNum,
                    void* param
);
```

Figure 3-38 FormatCallback Callback-Function

```
enum ODADriveSDK_FORMAT_METHOD
{
            ODADriveSDK_NO_BACKTRACK = 0,           ///< delete entries of files or directories only(volume
capacity will not be gained.)
            ODADriveSDK_BACKTRACK = 1               ///< erase used marker of discs (volume capacity
will be gained.)
};
```

Figure 3-39 FormatMethod Enum

```
/**
 * function: ODADriveSDK_DoAllDelete
 *
 * summary:
 *     Delete all files in the volume.
 *     If the medium is RE, a caller can select formatMethod either ODADriveSDK_BACKTRACK or
ODADriveSDK_NO_BACKTRACK.
 *     If the medium is WO, formatMethod is ignored.
 *     The callback function ODADriveSDK_DoAllDelete() is called only in case of formatMethod is
ODADriveSDK_BACKTRACK.
 *
 * param[in] drive:
 *                  Drive path of target
 *                  In case of Windows, set "G:", or "G:\" (case insensitive).
 *                  In case of Mac, set "/Volumes/XXX" or "/dev/diskX".
 *                  In case of Linux, set mount point.
 * param[in] formatMethod:
 *                  If ODADriveSDK_BACKTRACK, capacity of volume will be restored, but the deleted files will
never be
 *                  recoverable by roll back.(RE)
 *                  If ODADriveSDK_NO_BACKTRACK, capacity of volume will not be restored, but the deleted
file will be
 *                  recoverable by roll back. (RE/WO)
 * param[in] volumeType:
 *                  Volume type (refer ODADriveSDK_VOLUME_TYPE).
 *                  Valid only if formatMethod == ODADriveSDK_BACKTRACK. Otherwise, current
volumeType will be used.
 * param[in] callback:
 *                  Callback function for checking deleted disc num before format. (if not necessary, set NULL)
 * param[in] param:
 *                  User defined parameter for callback function. (if not necessary, set NULL)
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_DoAllDelete(
                    const ODADriveSDK_CHAR *drive,
                    enum ODADriveSDK_FORMAT_METHOD formatMethod,
                    enum ODADriveSDK_VOLUME_TYPE volumeType,
                    ODADriveSDK_AllDeleteCallback callback,
                    void *param
);
```

Figure 3-40 DoAllDelete Function

3.3.2.10 Finalize Write Once Media

An application can finalize the write once media by ODADriveSDK_DoFinalize(). However, the finalized medium becomes read-only forever, it is recommended for a lengthy storage life. This function works only for write once media.

After this API is completed, the volume is mounted on Windows and Macintosh OS X. On the other hand, the volume is unmounted on Linux.

```
/**
  * function: ODADriveSDK_DoFinalize
  *
  * summary:
  *     Finalyze the WO medium by this function.
  *     After finalizing, the cartridge will be read only.
  *
  * param[in] drive:
  *                     Drive path of target
  *                     In case of Windows, set "G:", or "G:\" (case insensitive).
  *                     In case of Mac, set "/Volumes/XXX" or "/dev/diskX".
  *                     In case of Linux, set mount point.
  *
  * return: zero if successful, or error code otherwise.
  */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_DoFinalize(
                        const ODADriveSDK_CHAR *drive
);
```

Figure 3-41 DoFinalize Function

3.3.2.11 Exclusive Access Mode (Windows only)

An application can enter exclusive access mode for the volume by ODADriveSDK_EnterExclusiveAccessMode(). When the application process enters exclusive access mode for the volume, any other process cannot open/create any files or directories except for /root directory. Only the caller application process which entering exclusive access mode has full access control for the volume.

The application can exit from exclusive access mode by either calling ODADriveSDK_ExitExclusiveAccessMode(), or close handle. Note that the handles possessed by the process will be closed automatically, when the process is terminated.

```
/**
 * function: ODADriveSDK_EnterExclusiveAccessMode
 *
 * summary:
 *     Enter the caller process to exclusive access mode,
 *
 * param[in] handle:
 *                  Pointer of volume HANDLE which has been opened by caller before this calling.
 *                  (e.g. by CreateFile() on Windows )
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_EnterExclusiveAccessMode(
                        void *handle
);
```

Figure 3-42   EnterExclusiveAccessMode Function

```
/**
 * function: ODADriveSDK_ExitExclusiveAccessMode
 *
 * summary:
 *     Exit the caller process from exclusive access mode,
 *
 * param[in] handle:
 *                  Pointer of volume HANDLE. The handle will be closed by caller after this calling.
 *                  (e.g. by CloseHandle() on Windows )
 *
 * return zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_ExitExclusiveAccessMode(
                        void *handle
);
```

Figure 3-43   ExitExclusiveAccessMode Function

3.3.2.12 Attributes in Cartridge Memory

The attributes in Cartridge Memory can be accessed by ODADriveSDK_EnumAttribute(), ODADriveSDK_ReadAttribute(), or ODADriveSDK_WriteAttribute().

```
struct ODA_DRIVE_SDK_CAPI ODADriveSDK_AttributeIdVector
{
        uint32_t attributeNum;                          ///< number of attributes
        uint16_t attributeIds[1];                   ///< attribute ids
};
```

Figure 3-44 AttributeIdVector Structure

```
/**
 * function: ODADriveSDK_EnumAttribute
 *
 * summary:
 *     Enumerate lists of attribute data in the cartridge memory.
 *
 * param[in] drive:
 *                  Drive path of target
 *                  In case of Windows, set "G:", or "G:\" (case insensitive).
 *                  In case of Mac, set "/Volumes/XXX" or "/dev/diskX".
 *                  In case of Linux, set mount point.
 * param[out] attributeIdBuffer:
 *                  Pointer to ODADriveSDK_AttributeIdVector (variable length)
 * param[in] attributeIdBufferSize:
 *                  Buffer size of attributeIdBuffer
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_EnumAttribute(
                const ODADriveSDK_CHAR *drive,
                struct ODADriveSDK_AttributeIdVector* attributeIdBuffer,
                uint64_t attributeIdBufferSize
);
```

Figure 3-45 EnumAttribute Function

```
/**
 * function: ODADriveSDK_ReadAttribute
 *
 * summary:
 *     Read attribute data from the cartridge memory.
 *
 * param[in] drive:
 *                      Drive path of target
 *                      In case of Windows, set "G:", or "G:\" (case insensitive).
 *                      In case of Mac, set "/Volumes/XXX" or "/dev/diskX".
 *                      In case of Linux, set mount point.
 * param[in] identifier:
 *                      Attribute identifier
 * param[out] flagsAndFormat:
 *                      Format[LSB0-1bit]={0(Binary),1(Ascii),2(Text)}, Reserved[2-6bit],
ReadOnly[7bit]={0(R/W),1(ReadOnly)}
 * param[out] value:
 *                      Value of attribute
 * param[in] valueBufferSize:
 *                      Buffer size of value
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_ReadAttribute(
                      const ODADriveSDK_CHAR *drive,
                      uint16_t identifier,
                      uint8_t* flagsAndFormat,
                      uint32_t* valueSizeReturned,
                      void* value, uint32_t valueBufferSize
);
```

Figure 3-46 ReadAttribute Function

```
/**
  * function: ODADriveSDK_WriteAttribute
  *
  * summary:
  *     Write attribute data to the cartridge memory.
  *
  * param[in] drive:
  *                     Drive path of target
  *                     In case of Windows, set "G:", or "G:\" (case insensitive).
  *                     In case of Mac, set "/Volumes/XXX" or "/dev/diskX".
  *                     In case of Linux, set mount point.
  * param[in] identifier:
  *                     Attribute identifier
  * param[in] flagsAndFormat:
  *                     Format[LSB0-1bit]={0(Binary),1(Ascii),2(Text)}, Reserved[2-6bit],
ReadOnly[7bit]={0(R/W),1(ReadOnly)}
  * param[in] value:
  *                     Value of attribute
  * param[in] valueBufferSize:
  *                     Buffer size of value
  *
  * return: zero if successful, or error code otherwise.
  */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_WriteAttribute(
                        const ODADriveSDK_CHAR *drive,
                        uint16_t identifier,
                        uint8_t flagsAndFormat,
                        const void* value,
                        uint32_t valueBufferSize
);
```

Figure 3-47 WriteAttribute Function

ID	Length	Format	Writable	Name
0003h	8	BINARY	No	LOAD COUNT
0004h	8	BINARY	No	MAM SPACE REMAINING
0007h	2	BINARY	No	INITIALIZATION COUNT
020Ah	40	ASCII	No	DEVICE VENDOR/SERIAL NUMBER AT LAST LOAD
020Bh	40	ASCII	No	DEVICE VENDOR/SERIAL NUMBER AT LAST LOAD-1
020Ch	40	ASCII	No	DEVICE VENDOR/SERIAL NUMBER AT LAST LOAD-2
020Dh	40	ASCII	No	DEVICE VENDOR/SERIAL NUMBER AT LAST LOAD-3
0224h	8	BINARY	No	LOGICAL POSITION OF FIRST ENCRYPTED BLOCK
0225h	8	BINARY	No	LOGICAL POSITION OF FIRST UNENCRYPTED BLOCK AFTER THE FIRST ENCRYPTED BLOCK
0400h	8	ASCII	No	MEDIUM MANUFACTURER
0401h	32	ASCII	No	MEDIUM SERIAL NUMBER
0406h	8	ASCII	No	MEDIUM MANUFACTURE DATE
0407h	8	BINARY	No	MAM CAPACITY
0408h	1	BINARY	No	MEDIUM TYPE
0409h	2	BINARY	No	MEDIUM TYPE INFORMATION
0800h	8	ASCII	No	APPLICATION VENDOR
0801h	32	ASCII	No	APPLICATION NAME
0802h	8	ASCII	No	APPLICATION VERSION
0803h	160	TEXT	No	USER MEDIUM TEXT LABEL

0804h	12	ASCII	No	DATE AND TIME LAST WRITTEN
0805h	1	BINARY	No	TEXT LOCALIZATION IDENTIFIER
0806h	32	ASCII	Yes	BARCODE
0D00h	1	BINARY	No	MAX SLOT NUMBER OF WRITTEN DISC
0D03h	1	BINARY	No	WRITE PROTECT
0D12h	24	BINARY	No	MEDIUM REWRITE COUNT
1100h	1	BINARY	No	CARTRIDGE TYPE
1101h	1	BINARY	No	DISC PACKAGING POLICY OF CARTRIDGE
1102h	1	BINARY	No	NUMBER OF DISCS IN CARTRIDGE
1500h	128	ASCII	No	VENDOR UNIQUE VOLUME INFORMATION
1501h	8	BINARY	No	TOTAL VOLUME SIZE
1502h	8	BINARY	No	REMAINING VOLUME SIZE
1503h	4	BINARY	No	TOTAL NUMBER OF FILES IN VOLUME
1504h	4	BINARY	No	TOTAL NUMBER OF DIRECTORIES IN VOLUME
1505h	256	ASCII	Yes	NDEF MESSAGE
1580h -159Bh	Max: 544	Not Specified	Yes	USER ATTRIBUTES

Table 3-1 Attributes

3.3.2.13 Allocate new disc

An application can allocate new disc to the volume by ODADriveSDK_AllocateNewDisc().

```
/**
 * function: ODADriveSDK_AllocateNewDisc
 *
 * summary:
 *     Change current writing disc to a next new disc.
 *     If there is any writing file handles on the volume, this calling will be failed.
 *     If no new disc in the cartridge, this callin will be failed.
 *
 * param[in] drive:
 *                     Drive path of target
 *                     In case of Windows, set "G:", or "G:\" (case insensitive).
 *                     In case of Mac, set "/Volumes/XXX" or "/dev/diskX".
 *                     In case of Linux, set mount point.
 *
 * return true if successful, or false otherwise.
 *
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_AllocateNewDisc(
                      const ODADriveSDK_CHAR *drive
);
```

Figure 3-48 AllocateNewDisc function

3.3.2.14 Flush volume buffers

An application makes all the volume buffers flush to the medium by ODADriveSDK_FlushVolumeBuffers().

There shall be no writable handle is opened when this function is called, otherwise the function will be failed by error.

```
/**
 * function: ODADriveSDK_FlushVolumeBuffers
 *
 * summary:
 *     Flush volume management data buffer to the disc forcibly.
 *
 * param[in]    drive:
 *                      Drive path of target
 *                      In case of Windows, set "G:", or "G:\" (case insensitive).
 *                      In case of Mac, set "/Volumes/XXX" or "/dev/diskX".
 *                      In case of Linux, set mount point.
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_FlushVolumeBuffers(
                        const ODADriveSDK_CHAR *drive
);
```

Figure 3-49 FlushVolumeBuffers Function

3.3.2.15 Raw mount flag (Windows only)

An application can set the raw mount flag to the volume by ODADriveSDK_SetRawMountFlag(). If the rawMountFlag is set (=1), the file system driver will complete mounting quickly, but will not provide normal file access (only the volume is able to be opened), and accessMode of volume will be ODADriveSDK_ACCESS_MODE_RAW from next mounting. If the rawMountFlag is not set (=0), the file system driver will mount normally as usual.

The rawMountFlag setting is remained during the system running, but will be reset at next system start.

To get current rawMountFlag setting, use ODADriveSDK_GetRawMountFlag().

To change the system behavior at the system starting, an application can set default raw mount flag by calling ODADriveSDK_SetDefaultRawMountFlag(). This default setting change shall take effect from next mounting, and shall be overridden by subsequent calling of ODADriveSDK_SetRawMountFlag().

To get current defaultRawMountFlag setting, use ODADriveSDK_GetDefaultRawMountFlag().

```
enum ODADriveSDK_RAW_MOUNT_FLAG
{
		ODADriveSDK_RAW_MOUNT_FLAG_OFF = 0, ///< raw mount flag off(default)
		ODADriveSDK_RAW_MOUNT_FLAG_ON = 1   ///< raw mount flag on
};
```

Figure 3-50 RawMountFlag Enum

```
/**
 * function: ODADriveSDK_SetDefaultRawMountFlag
 *
 * summary:
 *     Set default raw mount flag for the system.
 *     This flag setting is persitent and remained even after the system rebooted.
 *     This setting change shall take effect from next mounting,
 *     and shall be overridden by subsequent calling of ODADriveSDK_SetRawMountFlag().
 *
 * param[in] rawMountFlag:
 *                     Raw mount flag.
 *                     If ODADriveSDK_RAW_MOUNT_FLAG_OFF, raw mount flag is clear(default).
 *                     If ODADriveSDK_RAW_MOUNT_FLAG_ON, raw mount flag is set.
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_SetDefaultRawMountFlag(
		enum ODADriveSDK_RAW_MOUNT_FLAG defaultrawMountFlag
);
```

Figure 3-51 SetDefaultRawMountFlag Function

```
/**
 * function: ODADriveSDK_GetRawMountFlag
 *
 * summary:
 *     Get the default raw mount flag set by ODADriveSDK_SetDefaultRawMountFlag().
 *
 * param[out] defaultRawMountFlag:
 *                      Raw mount flag
 *                      If ODADriveSDK_RAW_MOUNT_FLAG_OFF, raw mount flag is clear(default).
 *                      If ODADriveSDK_RAW_MOUNT_FLAG_ON, raw mount flag is set.
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_GetDefaultRawMountFlag(
            enum ODADriveSDK_RAW_MOUNT_FLAG *defaultRawMountFlag
);
```

Figure 3-52 GetDefaultRawMountFlag Function

```
/**
 * function: ODADriveSDK_SetRawMountFlag
 *
 * summary:
 *     Set raw mount flag for the volume.
 *     The volume accessMode will be ODADriveSDK_ACCESS_MODE_RAW from next mount.
 *     This flag setting is remained during the system running, but will be reset
 *     at next system start.
 *
 * param[in] drive:
 *                      Drive path of target
 *                      In case of Windows, set "G:", or "G:\" (case insensitive).
 * param[in] rawMountFlag:
 *                      Raw mount flag.
 *                      If ODADriveSDK_RAW_MOUNT_FLAG_OFF, raw mount flag is clear(default).
 *                      If ODADriveSDK_RAW_MOUNT_FLAG_ON, raw mount flag is set.
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_SetRawMountFlag(
                        const ODADriveSDK_CHAR *drive,
                        enum ODADriveSDK_RAW_MOUNT_FLAG rawMountFlag
);
```

Figure 3-53 SetRawMountFlag Function

```
/**
  * function: ODADriveSDK_GetRawMountFlag
  *
  * summary:
  *     Get the volume raw mount flag set by ODADriveSDK_SetRawMountFlag().
  *     Note the current volume's accessMode will be obtained by ODADriveSDK_GetInformation().
  *
  * param[in] drive:
  *                     Drive path of target
  *                     In case of Windows, set "G:", or "G:\" (case insensitive).
  * param[out] rawMountFlag:
  *                     Raw mount flag
  *                     If ODADriveSDK_RAW_MOUNT_FLAG_OFF, raw mount flag is clear(default).
  *                     If ODADriveSDK_RAW_MOUNT_FLAG_ON, raw mount flag is set.
  *
  * return: zero if successful, or error code otherwise.
  */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_GetRawMountFlag(
                      const ODADriveSDK_CHAR *drive,
                      enum ODADriveSDK_RAW_MOUNT_FLAG *rawMountFlag
);
```

Figure 3-54 GetRawMountFlag Function

3.3.2.16 Remount volume (Windows only)

The volume will be remounted without cartridge ejecting by this calling. The setting of ODADriveSDK_SetRawMountFlag() will be reflect to the volume.

There shall be no handle is opened for the volume when this function is called, otherwise the function will be failed by error.

```
/**
 * function: ODADriveSDK_DoRemount
 *
 * summary:
 *     Remounted the volume without cartridge ejecting.
 *     Following API will be effective by this calling:
 *        ODADriveSDK_SetRawMountFlag()
 *
 * param[in] drive:
 *                       Drive path of target
 *                       In case of Windows, set "G:", or "G:\" (case insensitive).
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_DoRemount(
                          const ODADriveSDK_CHAR *drive
);
```

Figure 3-55 DoRemount Function

3.3.2.17 Current Loaded Disc

An application can get index number of the disc which is loaded in the internal optical drive unit by *ODADriveSDK_GetCurrentLoadedDiscIndex()*.

```
/**
  * function: ODADriveSDK_GetCurrentLoadedDiscIndex
  *
  * summary:
  *     Get index number of the disc which is loaded in the internal optical drive.unit
  *
  * param[in] drive:
  *                     Drive path of target
  *                     In case of Windows, set "G:", or "G:\" (case insensitive).
  *                     In case of Mac, set "/Volumes/XXX" or "/dev/diskX".
  *                     In case of Linux, set mount point.
  * param[out] index:
  *                     Current loaded disc index.
  *                     If ( 0 <= index < 12), current loaded disc index.
  *                     If ( index == 255 ), no disc is loaded.
  *                     Otherwise. reserved.
  *
  * return: zero if successful, or error code otherwise.
  */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_GetCurrentLoadedDiscIndex(
                        const ODADriveSDK_CHAR *drive,
                        uint8_t *index
);
```

Figure 3-56 GetCurrentLoadedDiscIndex Function

3.3.2.18 Volume label(Linux only)

An application can set the volume label by ODADriveSDK_SetVolumeLabel().

The usable characters for the volume label are from 1 to 63 characters in Unicode 2.0.

```
/**
 * function: ODADriveSDK_SetVolumeLabel
 *
 * summary:
 *     Set volume label.
 *
 * param[in]    drive:
 *                  Drive path of target
 *                  In case of Linux, set mount point.
 * param[in]    label:
 *                  New volume name
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_SetVolumeLabel (
                    const ODADriveSDK_CHAR *drive,
                    ODADriveSDK_CHAR *label
);
```

Figure 3-57 SetVolumeLabel Function

An application can get the volume label by ODADriveSDK_GetVolumeLabel().

```
/**
 * function: ODADriveSDK_GetVolumeLabel
 *
 * summary:
 *     Get volume label.
 *
 * param[in]    drive:
 *                  Drive path of target
 *                  In case of Linux, set mount point.
 * param[out]   label:
 *                  Volume name
 * param [in]   valueBufferSize:
 *                   buffer size of value.
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_GetVolumeLabel (
                    const ODADriveSDK_CHAR *drive,
                    ODADriveSDK_CHAR *label,
                    uint32_t valueBufferSize
);
```

Figure 3-58 GetVolumeLabel Function

3.3.2.19 Support Media Information

An application can get media information which the specified drive supports by ODADriveSDK_GetSupportedMediaInfo.

```
enum ODADriveSDK_SUPPORTED_MEDIA_STATUS
{
        ODADriveSDK_SUPPORTED_MEDIA_STATUS_READ_ONLY   = 0,
        ODADriveSDK_SUPPORTED_MEDIA_STATUS_READ_WRITE = 1
};
```

Figure 3-59 SupportedMediaStatus Enum

```
struct ODA_DRIVE_SDK_CAPI ODADriveSDK_SupportedMediaDescriptor
{
        char mediumManufacturer [8];
        char mediumProductIdentification [16];
        enum ODADriveSDK_SUPPORTED_MEDIA_STATUS mediumStatus;
        uint8_t        reserve[7];
};
```

Figure 3-60 Supported Media Descriptor Structure

```
struct ODA_DRIVE_SDK_CAPI ODADriveSDK_SupportedMediaInfo
{
        uint32_t sizeOfDescriptor
        struct ODADriveSDK_SupportedMediaDescriptor supportedMediaDescriptor[1];
};
```

Figure 3-61 Supported Media Info Structure

```
/**
 * function: ODADriveSDK_GetSupportedMediaInfo
 *
 * summary:
 *     Get Supported Media Information of a drive..
 *
 * param[in]    drive:
 *                      Drive path of target
 *                      In case of Windows, set "G:", or "G:\" (case insensitive).t
 *                      In case of Mac, set "/Volumes/XXX", "/dev/diskX", or "deviceId".
. *                       In case of Linux, set mount point, "/dev/sdX", or"/dev/sgX". * param[out]    label:
 *                      Volume name
 * param [out]    supportedMediaInfoBuffer:
 *                       Pointer to ODADriveSDK_SupportedMediaInfo (variable length)
 *
 * param [in]    supportedMediaInfoSize:
 *                       Buffer size of supportedMediaInfoBuffer
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_GetSupportedMediaInfo (
                         const ODADriveSDK_CHAR *drive,
                        struct ODADriveSDK_SupportedMediaInfo* supportedMediaInfoBuffer,
                        uint64_t supportedMediaInfoSize
);
```

Figure 3-62 GetSupportedMediaInfo Function

3.3.3 File operations

3.3.3.1 File Recording Information

An application can obtain each file’s recording information in the discs in the cartridges by ODADriveSDK_GetFileRecordingInfo().

```
/** file recording information */
struct ODA_DRIVE_SDK_CAPI ODADriveSDK_FileRecordingInfo
{
        uint32_t discId;      ///< recording disc id(equal to disc index)
        uint64_t fileSize; ///< recording file offset in the disc (Byte)
};

/** file recording information vector */
struct ODA_DRIVE_SDK_CAPI ODADriveSDK_FileRecordingInfoVector
{
        uint32_t infoNum;  ///< number of ODADriveSDK_FILE_RECORDING_INFO
        struct ODADriveSDK_FileRecordingInfo infos[1];            ///< file recording information
};
```

Figure 3-63 FileRecordingInfoVector Structure

```
/**
 * function: ODADriveSDK_GetFileRecordingInfo
 *
 * summary:
 *     Get recording information of the file.
 *     The recording information is consist of {disc id, size}
 *     example:
 *        If a 20GiB file is recorded 10GiB each discs in Disc1, Disc2,
 *        this API return fileRecordingInfo such as {disc1, 10GiB}, {disc2, 10GiB}.
 *
 * param[in] filePath:
 *                    File path of target
 * param[out] fileRecordingInfoBuffer:
 *                    File recording information
 * param[in] fileRecordingInfoBufferSize:
 *                    Fuffer size of fileRecordingInfoBuffer
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_GetFileRecordingInfo(
                        const ODADriveSDK_CHAR *filePath,
                        struct ODADriveSDK_FileRecordingInfoVector* fileRecordingInfoBuffer,
                        uint64_t fileRecordingInfoBufferSize
);
```

Figure 3-40 GetFileRecordingInfo Function

3.3.3.2 File Control Option

An application can set the control options of writing file by ODADriveSDK_SetFileControlOption () or ODADriveSDK_SetFileControlOptionEx ().

ODADriveSDK_SetFileControlOption() is obsolete function. For new application development, use the ODADriveSDK_SetFileControlOptionEx () instead.

```
enum ODADriveSDK_FILE_CONTROL_OPTION_OF_FS_FLUSH
{
            ODADriveSDK_REFRAIN_FS_FLUSH_AT_CLOSE = 0, ///< refrain FS flush at close
            ODADriveSDK_FORCE_FS_FLUSH_AT_CLOSE = 1     ///< force FS flush at close
};
```

Figure 3-64 FileControlOptionOfFsFlush Enum

```
/**
 * function: ODADriveSDK_SetFileControlOption
 *
 * summary:
 *     Set control option to writing file handle,
 *     This function is provided for backward compatibility,
 *     but obsolete.
 *
 * param[in] handle:
 *                 Depends on OS.( Win: pointer of opened HANDLE. Linux: pointer of opened file descriptor)
 * param[in] fsFlushOption:
 *                 File control options of FS flush (refer
ODADriveSDK_FILE_CONTROL_OPTION_OF_FS_FLUSH).
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_SetFileControlOption(
                        void *handle,
                        enum ODADriveSDK_FILE_CONTROL_OPTION_OF_FS_FLUSH fsFlushOption
);
```

Figure 3-65 SetFileControlOption Function

```
#define ODADriveSDK_FILE_INHIBIT_SPANNING_DISC_WRITE          0x00000002
#define ODADriveSDK_FILE_REFRAIN_FS_FLUSH_AT_CLOSE            0x00000004
#define ODADriveSDK_FILE_FORCE_FS_FLUSH_AT_CLOSE              0x00000008
#define ODADriveSDK_FILE_PACKED_WRITE                         0x00000010
```

Figure 3-66 FileControlOptionFlag

```
/**
 * function: ODADriveSDK_SetFileControlOptionEx
 *
 * summary:
 *     Set control option to writing file handle,
 *
 * param[in] handle:
 *                  Depends on OS.( Win: pointer of opened HANDLE. Linux: pointer of opened file descriptor)
 * param[in] fileControlOptionFlag:
 *                  File control options flag(refer file control option flag).
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_SetFileControlOptionEx(
            void *handle,
            uint32_t fileControlOptionFlag
);
```

Figure 3-67 SetFileControlOptionEx Function

Name of flag	**Remarks**
ODADriveSDK_FILE_INHIBIT_ SPANNING_DISC_WRITE	**If this flag is set, the file been writing by this handle will not be able to span the disc even when the written disc is out of space. When the written disc is out of space the write function of the handle will be failed by error.** **If ODADriveSDK_FILE_PACKED_WRITE is also set, this flag will be ignored.**
ODADriveSDK_FILE_FORCE_ FS_FLUSH_AT_CLOSE	**If this flag is set, the FS buffer will been flushed when the file handle is closed.** **This flag will override ODADriveSDK_FILE_REFRAIN_FS_FLUSH_AT_CLOSE or ODADriveSDK_FILE_PACKED_WRITE if those flags are also set.**
ODADriveSDK_FILE_REFRAIN _FS_FLUSH_AT_CLOSE	**If this flag is set, the FS buffer will not been flushed even when the file handle is closed.** **This flag will be ignored if ODADriveSDK_FILE_FORCE_FS_FLUSH_AT_CLOSE flag is also set.** **This flag will override ODADriveSDK_FILE_PACKED_WRITE.**

ODADriveSDK_FILE_PACKED _WRITE	**If this flag is set, the written file of the handle will been buffered on memory even after the file handle is closed. This flag should be set to speed up series of small file writing.** **This flag will be ignored If ODADriveSDK_FILE_REFRAIN_FS_FLUSH_AT_CLOSE or ODADriveSDK_FILE_FORCE_FS_FLUSH_AT_CLOSE are also set.** **See Detect Write Error for error checking.**

Table 2: fileControlOptionFlags of SetFileControlOptionEx

3.3.3.3 Hash Information

An application can add the hash information to a file by ODADriveSDK_AddHashINfo ().An application can also obtain the hash information of a file by ODADriveSDK_GetHashInfo() and remove the hash information of a file by ODADriveSDK_RemoveHashInfo() .

```
/**   hash type */
enum ODADriveSDK_HASH_TYPE
{
            ODADriveSDK_HASH_TYPE_NON    = 0,        ///< Hash has not been set
            ODADriveSDK_HASH_TYPE_MD5    = 3         ///< MD5
};
```

Figure 3-68 Hash Type

```
/** hash info */
struct ODA_DRIVE_SDK_CAPI ODADriveSDK_HashInfo
{
            enum ODADriveSDK_HASH_TYPE hashType;            ///< hash type
            uint16_t                    hashLength;         ///< size of hash value
            uint8_t                     hashValue[1];       ///< hash value
};
```

Figure 3-69 Hash Infomation

```
/**
 * function: ODADriveSDK_GetHashInfo
 *
 * summary:
 *     Get Hash Info of a file in the volume.
 *
 * param[in] filePath:
 *                     File path of target
 * param[out] hashInfoBuffer
 *                       Pointer to ODADriveSDK_HashInfo (variable length)
 * param[in] hashInfoBufferSize
 *                       Buffer size of hashInfoBuffer
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t ODADriveSDK_GetHashInfo(const ODADriveSDK_CHAR *filePath,
struct ODADriveSDK_HashInfo* hashInfoBuffer, uint64_t hashInfoBufferSize);
```

Figure 3-70 GetHashInfomation Function

```
/**
 * function: ODADriveSDK_AddHashInfo
 *
 * summary:
 *     Add Hash Info to a file in the volume.
 *
 * param[in] filePath:
 *                    File path of target
 * param[in] hashInfoBuffer
 *                     Pointer to ODADriveSDK_HashInfo (variable length)
 * param[in] hashInfoBufferSize
 *                     Buffer size of hashInfoBuffer
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t ODADriveSDK_AddHashInfo(const ODADriveSDK_CHAR *filePath,
struct ODADriveSDK_HashInfo* hashInfoBuffer, uint64_t hashInfoBufferSize);
```

Figure 3-71　AddHashInfomation Function

```
/**
 * function: ODADriveSDK_RemoveHashInfo
 *
 * summary:
 *     Remove Hash Info of a file in the volume.
 *
 * param[in] filePath:
 *                    File path of target
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t ODADriveSDK_RemoveHashInfo(const ODADriveSDK_CHAR *filePath);
```

Figure 3-72　RemoveHashInfomation Function

3.3.3.4 Direct read access (Windows only)

An application can read file by ODADriveSDK_GetFileAllocationInfo/ReadFile/CloseFile with using file allocation which is obtained by ODADriveSDK_GetFileAllocationInfo.

The file can be opened whether the volume mounted normally or raw by this calling.

The start position of read (parameter *whence*) and size (parameter *requestSize*) shall be aligned to 65,536 bytes for ODADriveSDK_ReadFile().

Only one file can be opened at the time. Therefore, the caller shall close the opened handle by ODADriveSDK_CloseFile() before calling next ODADriveSDK_OpenFileHandleWithAllocationInfo().

```
/** file allocation information */
struct ODA_DRIVE_SDK_CAPI ODADriveSDK_FileAllocationInfo
{
        uint32_t discId;            ///< recording disc id(equal to disc index)
        uint32_t blockOffset;       ///< recording block offset [2,048bytes/block]
        uint32_t numberOfBlocks;    ///< number of blocks of this allocation
};

/** file recording information vector */
struct ODA_DRIVE_SDK_CAPI ODADriveSDK_FileAllocationInfoVector
{
        uint64_t opaque;            /// opaque value for caller, but shall be saved.
        uint64_t fileSize;          /// size of the file
        uint32_t infoNum;           ///< number of ODADriveSDK_FILE_ALLOCATION_INFO
        struct ODADriveSDK_FileAllocationInfo infos[1];       ///< file allocation information
};
```

Figure 3-73 FileAllocationInfoVector Structure

```
/**
 * function: ODADriveSDK_GetFileAllocationInfo
 *
 * summary:
 *     Get allocation information of the file.
 *     The allocation information can be used for ODADriveSDK_OpenFileWithAllocationInfo().
 *
 * param[in]   filePath:
 *                 File path of target
 * param[out] fileAllocationInfoBuffer:
 *                 File allocation information
 * param[in]   fileAllocationInfoBufferSize:
 *                 Buffer size of fileAllocationInfoBuffer
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_GetFileAllocationInfo(
                    const ODADriveSDK_CHAR *filePath,
                    struct ODADriveSDK_FileAllocationInfoVector* fileAllocationInfoBuffer,
                    uint64_t fileAllocationInfoBufferSize
);
```

Figure 3-74 GetFileAllocationInfo Function

```
/**
 * function: ODADriveSDK_OpenFileWithAllocationInfo
 *
 * summary:
 *     Open file with allcationwhich is obtained by ODADriveSDK_GetFileAllocationInfo().
 *     File can be opened whether the volume is mounted normally or raw.
 *     Only one file handle can be opened at the time. It means caller shall close the
 *     opened file handle by ODADriveSDK_CloseFile() before calling next
ODADriveSDK_OpenFileWithAllocation().
 *
 * param[in]   drive:
 *                 drive path of target
 *                 In case of Windows, set "G:", or "G:\" (case insensitive).
 * param[in]   fileAllocationInfoBuffer:
 *                 File allocation information
 * param[in]   fileAllocationInfoBufferSize:
 *                 Buffer size of fileAllocationInfoBuffer.
 * param[out] handle:
 *                 Depends on OS.( Win: pointer of opened HANDLE)
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_OpenFileWithAllocationInfo(
                    const ODADriveSDK_CHAR *drive,
                    const struct ODADriveSDK_FileAllocationInfoVector* fileAllocationInfoBuffer,
                    uint64_t fileAllocationInfoBufferSize,
                    void *handle
);
```

Figure 3-75 OpenFileWithAllocationInfo Function

```
/**
 * function: ODADriveSDK_ReadFile
 *
 * summary:
 *     Read the file data from ODA to the buffer by this function
 *     The file handle shall have been opened by ODADriveSDK_OpenFileWithAllocationInfo().
 *     The read request shall be aligned to 64KiB(0x10000).
 *
 * param[in]    handle:
 *                  File handle opened by ODADriveSDK_OpenFileWithAllocationInfo().
 * param[in]    whence:
 *                  Start position to read file [bytes].
 *                  If value < 0(minus value), use the read pointer which holds last position of reading done.
 *                  Otherwise(zero or larger), shall be aligned to 65536(0x10000).
 * param[in]    requestSize:
 *                  Request size to read file [bytes].
 *                  Shall be aligned to 65536(0x10000), and 67108864(0x4000000) at maximum.
 * param[out] buffer:
 *                  Buffer to read data in.
 *                  Caller shall prepare the buffer which size is equal or larger than requestSize above.
 * param[out] resultSize:
 *                  Result size that the buffer has been filled with actual read data.
 *
 * return: zero if successful, or error code otherwise.
*/
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_ReadFile(
                    void *handle, int64_t whence,
                    uint32_t requestSize,
                    void* buffer, uint32_t *resultSize
);
```

Figure 3-76 ReadFile Function

```
/**
 * function: ODADriveSDK_CloseFile
 *
 * summary:
 *     Close a file handle opened by ODADriveSDK_OpenFileWithAllocationInfo().
 *
 * param[in]    handle:
 *                  File handle opened by ODADriveSDK_OpenFileWithAllocationInfo().
 *
 * return: zero if successful, or error code otherwise.
 */
ODA_DRIVE_SDK_CAPI uint64_t
ODADriveSDK_CloseFile(
                    void *handle
);
```

Figure 3-77 OpenFileWithAllocationInfo Function

4 Guideline

4.1 Detect ODA drives

4.1.1 Windows

The application calls GetLogicalDrives() to retrieve bitmask representing the currently available disk drives. Next the application calls ODADriveSDK_GetDeviceInformation() for each drive letters according to the bitmask of available drives above, and checks the return value of the function. If ODADriveSDK_GetDeviceInformation() returns success (zero), it is ODA drive otherwise not ODA drive.

4.1.2 Macintosh OSX

The application can call ODADriveSDK_DriveIdVector to get lists of ODA drive identifiers.

4.1.3 Linux

The application can call ODADriveSDK_GetDeviceInformation() for each SCSI generic device files such as /dev/sdX and /dev/sgX, and checks the return value of the function. If ODADriveSDK_GetDeviceInformation() returns success (zero), it is ODA drive otherwise not ODA drive.

4.2 DIRECT READ ACCESS (Advanced)

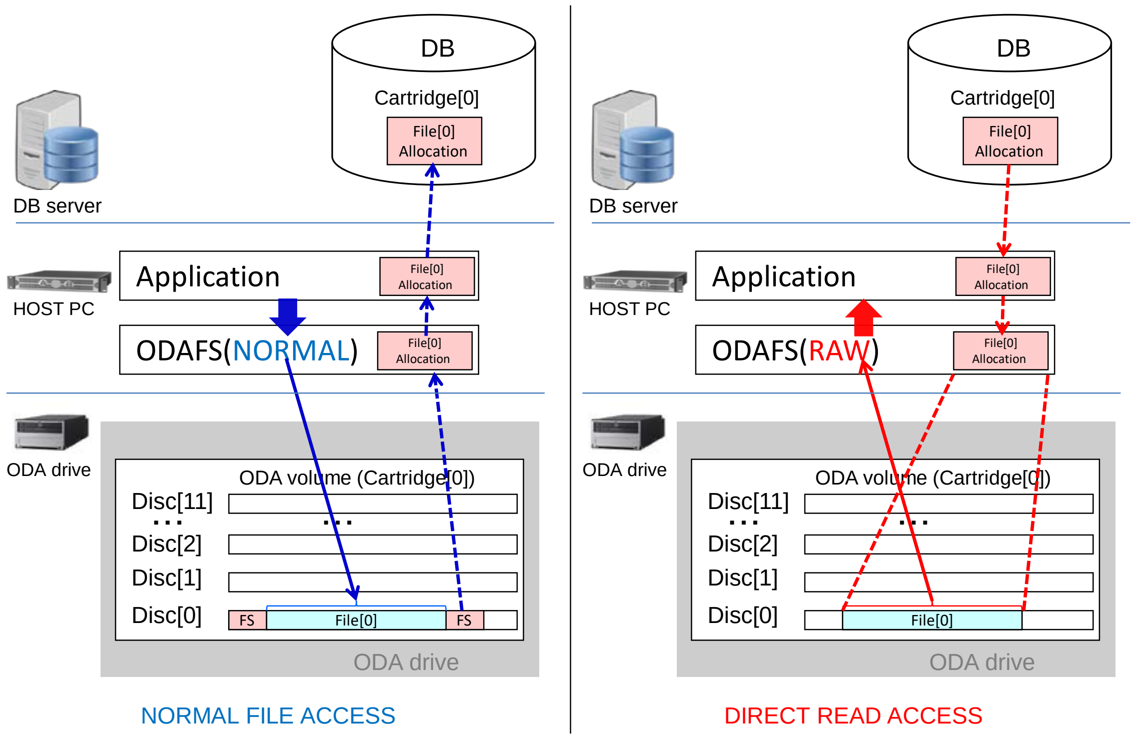


Figure 4-1 NORMAL FILE ACCESS and DIRECT READ ACCESS

ODAFS driver ver 3.2.0 or later supports direct read access feature to reduce the time of retrieving file from when the cartridge is located outside of a drive. This direct read access feature is assumed for applications which supports ODA library (ODS-L30M Petasite).
ODAFS driver ver 3.2.0 or later also supports disc auto load disabling feature. It may save 20 to 30 seconds constantly for FS mounting. When it works, ODA drive will not load the disc which FS management data is recorded on, and ODAFS driver will parse the FS management data on local HDD cache instead. This disc auto load disabling feature will work in the background. Therefore, an application can not control this feature.
On the other hand, the direct read access feature can be controlled from an application, and it will be able to reduce the parsing time of FS management data of ODAFS driver. The FS parsing time depends on the number of files and PC performance, and it may cost 90-120 seconds at maximum.
The direct read access feature saves the FS parsing time to 0.1 seconds, because ODAFS driver will not parse the FS management data. However, an application can read the file via ODA Drive SDK.

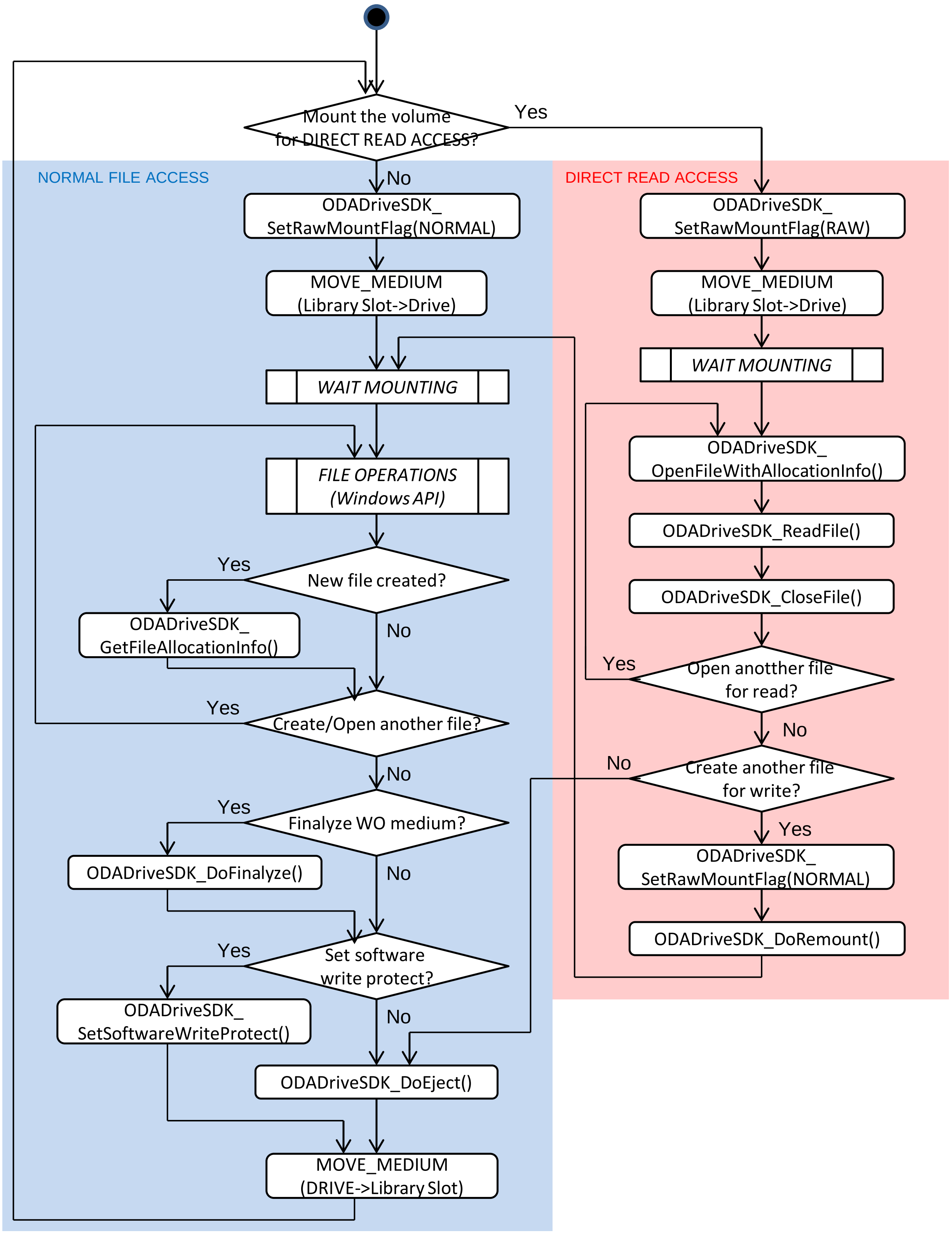

Mount the volume for DIRECT READ ACCESS?
Yes
No
NORMAL FILE ACCESS
DIRECT READ ACCESS
ODADriveSDK_ SetRawMountFlag(NORMAL)
MOVE_MEDIUM (Library Slot->Drive)
WAIT MOUNTING
FILE OPERATIONS (Windows API)
New file created?
ODADriveSDK_ GetFileAllocationInfo()
Create/Open another file?
Finalyze WO medium?
ODADriveSDK_DoFinalyze()
Set software write protect?
ODADriveSDK_ SetSoftwareWriteProtect()
ODADriveSDK_DoEject()
MOVE_MEDIUM (DRIVE->Library Slot)
ODADriveSDK_ SetRawMountFlag(RAW)
MOVE_MEDIUM (Library Slot->Drive)
WAIT MOUNTING
ODADriveSDK_ OpenFileWithAllocationInfo()
ODADriveSDK_ReadFile()
ODADriveSDK_CloseFile()
Open anotther file for read?
Create another file for write?
ODADriveSDK_ SetRawMountFlag(NORMAL)
ODADriveSDK_DoRemount()

Figure 4-2 Flowchart for support direct read access

Above Figure 4-1 NORMAL FILE ACCESS and DIRECT READ ACCESS shows how direct read access feature works. And Figure 4-2 Flowchart for support direct read access shows how an application which supports direct read access should use ODA Drive SDK. Applications which support ODA library are assumed, so that it moves a medium by MOVE MEDIUM of ODS-L30M's SCSI Media Changer command.
The application will write files by normal manner like left diagram in the former figure. ODAFS is mounted normally at that time and the application writes or reads files via standard I/F. After writing a file, the application shall get file allocation information from the written files via ODADriveSDK_GetFileAllocation(), and store the file allocation information to its database related to the written file. The application will eject the medium after when it finishes file access.
Only for the retrieving files, the application set raw mount flag to the volume before injecting the medium to drive by ODADriveSDK_SetRawMountFlag(RAW) like right diagram in the former figure. By this setting, ODAFS will be mounted raw from next mounting without FS parsing. In spite of ODAFS does not provide normal file access as raw mount, the application can open the file with the file allocation information loaded from its database by ODADriveSDK_OpenFileWithAllocationInfo(). After that, the application can read the portion or entire file by ODADriveSDK_ReadFile().
The application can change the volume mount from raw to normal by ODADriveSDK_SetRawMountFlag() and ODADriveSDK_DoRemount() without ejecting the medium.